SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered) UNCLASSIFIED

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER
2. GOVT ACCESSION NO.
3. RECIPIENT'S CATALOG NUMBER

4. TITLE (and Subtitle)

MINIMIZING THE NUMBER OF CLUSTERS IN MOBILE PACKET RADIO NETWORKS

5. TYPE OF REPORT & PERIOD COVERED
Thesis

6. PERFORMING ORG. REPORT NUMBER
LIDS-TH-1502

AUTHOR(s)

Abhay Kumar Parekh

8. CONTRACT OR GRANT NUMBER(s)
DARPA Order No. 3045/2-2-84
Amendment #11
ONR/N00014-84-K-0357

PERFORMING ORGANIZATION NAME AND ADDRESS
Massachusetts Institute of Technology
Laboratory for Information and Decision Systems
Cambridge, Massachusetts 02139

10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
Program Code No. 5T10
ONR Identifying No. 049-383

CONTROLLING OFFICE NAME AND ADDRESS
Defense Advanced Research Projects Agency
1400 Wilson Boulevard
Arlington, Virginia 22209

12. REPORT DATE
October 1985

13. NUMBER OF PAGES
83

MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)
Office of Naval Research
Information Systems Program
Code 437
Arlington, Virginia 22217

15. SECURITY CLASS. (of this report)
UNCLASSIFIED

15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release: distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

epsilon

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

Linked-Cluster Architectures have been suggested in the literature for organizing the radios of a stationless mobile Packet Radio Networ (PRN). Existing algorithms for achieving such architectures do not attempt to minimize the number of clusters and gateway nodes, aims which we claim are essential to the implementation of any Multiple Access scheme. The problem is formulated on graphs in 3 different ways, all of which are NP-complete. It is also shown that ε-Polynomial Time Algorithms are not likely to exist. A simple centralized heuristic, Greedy is analyzed in terms of its worst case fractional error. It is then argued that no efficient

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-LF-014-6601

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

20. (Continued)
heuristic is likely to be significantly better in terms of worst-case performance Two Distributed Linked-Cluster Algorithms are presented. The first, GLCA minimizes the number of clusters identically to Greedy. The communication complexity, C, of this algorithm ignoring collision is $O(N^2-N\sqrt{2M+1})$ for sparse graphs and $O(MN - M\sqrt{2M+1})$ for both dense and sparse graphs. The second, Tree Linked Cluster Algorithm (TLCA) runs in $C = O(M)$ and $T = O(N)$ (also ignoring collisions). The number of clusters is not minimized as well as GCLA, but communication costs are significantly lower. Schemes for collision minimization and resolution are suggested for both algorithms.

MINIMIZING THE NUMBER OF CLUSTERS IN MOBILE PACKET RADIO NETWORKS

by

Abhay Kumar Parekh

B.E.S. Johns Hopkins University
(December 1983)

SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE
DEGREE OF

MASTER OF SCIENCE
IN OPERATIONS RESEARCH

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 1985

Signature of Author Abhay K. Parekh
August 4, 1985

Certified by Robert G. Gallager
Robert G. Gallager
Thesis Supervisor

Accepted by
Jeremy F. Shapiro
Co-Director, Operations Research Center

MINIMIZING THE NUMBER OF CLUSTERS IN MOBILE PACKET RADIO NETWORKS

by

Abhay Kumar Parekh

SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE
DEGREE OF
MASTER OF SCIENCE
IN OPERATIONS RESEARCH
at the
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
September 1985

ABSTRACT

Linked-Cluster Architectures have been suggested in the literature for organizing the radios of a stationless mobile Packet Radio Network (PRN). Existing algorithms for achieving such architectures do not attempt to minimize the number of clusters and gateway nodes, aims which we claim are essential to the implementation of any Multiple Access scheme. The problem is formulated on graphs in 3 different ways, all of which are NP-complete. It is also shown that ϵ-Polynomial Time Algorithms are not likely to exist. A simple centralized heuristic, *Greedy* is analyzed in terms of its worst case fractional error. It is then argued that no efficient heuristic is likely to be significantly better in terms of worst-case performance. Two Distributed Linked-Cluster Algorithms are presented. The first, *GLCA* minimizes the number of clusters identically to *Greedy*. The communication complexity, C, of this algorithm ignoring collision is $O(N^2 - N\sqrt{2M+1})$ for sparse graphs and $O(MN - M\sqrt{2M+1})$ for dense graphs with N nodes and M edges. The time complexity, T, is $O(N - \sqrt{2M+1})$ for both dense and sparse graphs. The second, Tree Linked Cluster Algorithm (*TLCA*) runs in $C = O(M)$ and $T = O(N)$ (also ignoring collisions). The number of clusters is not minimized as well as in *GCLA*, but communication costs are significantly lower. Schemes for collision minimization and resolution are suggested for both algorithms.

Thesis Supervisor: Prof. Robert G. Gallager
Title: Professor of Electrical Engineering and Computer Science.

ACKNOWLEDGEMENTS

I would like to thank Robert Gallager for supervising this thesis with care, consideration and interest. It has been a pleasure working for him.

Thanks are due to Prof. David Shimoys for giving me a good overview of Approximate Algorithms, and providing the motivation for the proof of Theorem 2.2. I am sure the reader will join me in thanking Art Giordani for his fine artwork. Thanks also to Patrick Hosein for helping with the graphs of Figure 2.7.

The genial atmosphere of the Laboratory for Information and Decision Systems, and the numerous conversations I have had with its graduate students are much appreciated.

There have many people who have made my stay at MIT a very enjoyable one. I thank them all for the interesting discussions we have had on subjects unrelated to Technology and Engineering. Dicki Munsif, despite being so far away, gave me all the love, understanding and friendship that could ever be desired.

Finally, and most importantly I would like to acknowledge the immeasurable contribution of my parents. Their love, advice and sacrifices have made everything possible. This thesis is dedicated to them with the utmost love and respect.

TABLE OF CONTENTS

TABLE OF FIGURES

CHAPTER I
INTRODUCTION

A Packet Radio Network (PRN) is a communications network in which a set of geographically distributed, possibly mobile computers communicate over a shared broadcast medium (called the access channel.) The network is packet switched to facilitate the efficient management of bursty user traffic. The concept of bypassing telephone lines in favor of space as the transmitting medium for packet networks originated at the University of Hawaii in the early 1970's. Their so-called ALOHA system consisted of a single omnidirectional antenna which served as a hub, or station for all terminal communication. The radio channel was divided into two: a multiaccess channel, and a broadcast channel which was used by the central computer to communicate with the terminals.

The success of the ALOHA network stimulated extensive research in the area, and the model was extended to suit a variety of tactical and civilian situations, some of which are are listed below:

(1) Networking mobile radios in the battlefield [1].

(2) Communication in a Naval Battlegroup [5].

(3) Commercial applications in highly uneven terrain [1].

(4) Cellular Mobile Telephones [12].

Notice that these applications require that the users be *possibly mobile*, which intro-

duces the problem of designing protocols for networks whose topology is subject to change. Another issue which is immediately raised is the effective utilization of the access channel when all the terminals are not within line of sight of each other. Suggested future applications for PRN's bank critically on the assumption that these problems have been, or can be solved. For example, Nelson has noted the suitability of PRN's to establish communications following a natural disaster such as an earthquake, because of their "flexible topologies"[3]; Kahn has refered to "personal terminals" which could be carried about by individuals allowing them to communicate, via a PRN, with other mobile users, and with a variety of computer resources[2]; and Licklider[4] has considered a rather inventive application in which the physical condition of (mobile) elderly persons, equipped with various sensor devices, is continuously monitored by a home computer through a PRN.

In some of these applications, such as Cellular Mobile Telephones, it is realistic to assume that the network can be controlled by stations, which serve as fixed reference points in the otherwise mobile network. Much progress has been made on this front, as is clear from the rapidly growing use of Mobile Telephones. However, in many military scenarios, such as networking radios in a battlefield, one cannot assume the feasiblity of building stations. This, coupled with the fact that mutual line of sight among all the radios cannot be assured, complicates the problem of designing efficient protocols for network organization and mangagement, and presents to the PRN designer a challenge she has not as yet met successfully.

The next few sections serve to justify the previous statement, and to aquaint the reader briefly, with some of the design issues of Mobile PRN's. These sections will also allow us to present the specific problem the rest of thesis will address, and to present the model it will use to do this. For good surveys of PRN's see [1] and [2].

1.1. Technical System Considerations:

Three technical constraints that apply to the design of any PRN are outlined. Since these constraints are technology dependent ones, we emphasise their importance here, but do not dwell on them at any length in subsequent sections.

(a) Propogation Loss: It has been found that the propogation loss of ground radios is highly sensitive to the type of terrain the PRN is located in. This makes the estimation of link connectivity very hard when undulations in the terrain are not easy to predict, and strengthens the case for finding automated network management procedures capable of sensing the connectivity of the PRN in real-time.

(b) Multipath effects: These are reflections of the original signal which lead to superposition at the receiver of several copies of the signal. This causes symbol interference and fading, and consequently drives up the error rate. Movement by a mobile user of only a few meters can cause the received signal strength to drop below the threshold, effectively disabling the link. It then becomes possible for a link to be intermittently disabled and enabled even if the user is moving about within a radius of only a few meters. These effects are minimized by using spread-spectrum signalling, which is a combination of bandwidth expansion and coding. It is assumed in this thesis that multipath effects are negligible.

(c) Capture Effects: A packet is said to be captured at the receiver when all successive interfering packets at that receiver are rejected as noise. Unfortunately, it is possible that a later arriving signal with a considerably greater strength might destroy a captured packet. It is not clear how to model this situation exactly (eg. beyond what threshold is a captured packet actually destroyed?), and so we make the worst case assumption that all colliding packets at the receiver are destroyed.

For example, consider the situation in Fig. 1.1. Every radio has a fixed transmission

radius and all radios within this radius will hear it. Let A be transmitting to D. Clearly B can hear the broadcast. Now suppose that C would like to transmit to B. Then some packets from C are sure to collide with those from A, at the node B.

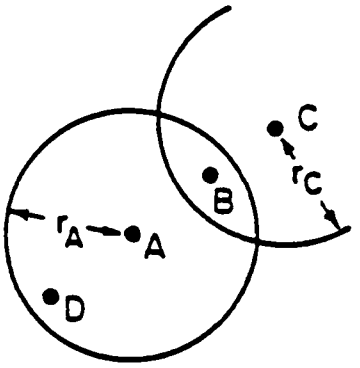


Fig. 1.1
Capture Assumption: r_A = transmission radius of A; r_C = transmission radius of C.

We assume that all packets from C and A transmitted at overlapping time intervals will be destroyed at node B.

It is worth observing here that in fixed topology PRN's, the transmission radii can be fixed to an optimum so that the effects of collisions are minimized, and throughput is maximized. However in a mobile environment, the network topology may be varying quite rapidly, and so this approach will not work. We therefore assume the transmission radii to be *arbitrary and fixed*.

1.2. Multiaccess Methods and the Hidden Terminal Problem

The throughput of a broadcast network depends critically on the method its users employ to access the channel. It is clear that if all users were to broadcast at random the large number of collisions would result in a poor level of thoughput. Many schemes have been suggested for broadcast networks in general. Carrier sensing methods have been found to be particularly attractive. By carrier sensing we refer to protocols in which the terminal listens to the channel to check if another terminal is broadcasting, and only transmits a packet when the channel is sensed to be idle. An important assumption made in the analysis of these schemes is that the terminals are within mutual line of sight (los), and within range of each other. Unfortunately, this does not apply to many PRN's. In fact it is possible for two terminals to be within los but not within range of each other, or for them to be obstructed by an object which blocks radio waves, such as a hill or a tall building. Such terminals are said to be hidden from each other. It has been shown that hidden terminals seriously degrade the throughput performances of traditional carrier sensing methods[6]. Intuitively this effect can be explained by observing that when a terminal senses the channel, it has no way to determine if a hidden terminal is broadcasting at the time.

Tobagi and Kleinrock [6] have suggested a *Busy Tone* solution to this problem by proposing the location of a central station such that all terminals are within los and mutual range of it. The station transmits a Busy tone on a specially set up busy-tone channel to all terminals, as long as it senses a transmission on the multiaccess channel. The terminals transmit only when they do not sense the busy tone. It has been shown that this solution is able to compensate substantially, for hidden terminal effects. A fixed station is also useful in cases when non-random access to the channel is desired such as polling, or centralized reservations, which suggests that whenever such controllers may be constructed economically, they should be. However, as noted earlier, many important applications of PRN's are in necessarily stationless environments, and it is not clear how to resolve the

Hidden terminal problem in such situations.

1.3. Linked Cluster Architectures

The ideas in the previous section provide the motivation for this scheme, which was first proposed by Baker and Ephremides [5]. It is reasonable to try to enhance the conflict resolution capability of a stationless PRN by organizing it into clusters, each of which has a node designated as the leader or head of that cluster. This node is bidirectionally connected to every other node in the cluster, and acts as a local station or controller. [1] Each node is in some cluster, and therefore is controlled by some cluster head. The hidden terminal problem is solved within a cluster using Busy-Tone Multiple Access or some non-random scheme such as polling.

Observe that in general it is neccessary to have more than one cluster in a network, since the transmission radii of the radios are fixed, and it is possible that no radio is connected to all the other nodes. This brings up the problem of inter-cluster communication, which is handled by linking adjacent cluster leaders through other nodes called gateway nodes. A gateway node is necessary if two adjacent cluster leaders are not within range of each other. Thus the organization of the PRN consists of possibly overlapping clusters, whose leaders form a backbone network, and are linked by gateway nodes. It should be understood that since the nodes are mobile, any node may become a cluster leader, or a gateway node, and that the status of a node will in general, keep changing. An algorithm which organizes the nodes of a PRN in this manner must not depend on the existence of a particular node in a particular region, thus forcing it to be distributed in nature.

[1] If node i can hear node j then we say that j is accessible from i. From now on two nodes are defined to be connected iff they are accessible from each other.

1.4. The Case For Minimizing The Number of Clusters

The major difference between the solutions proposed for the Hidden Terminal problem in sections 1.2 and 1.3 is that there are *many* controllers in the stationless environment. This leads to complications which arise from the existence of gateway nodes, and the possibility of one node being contained in several clusters.

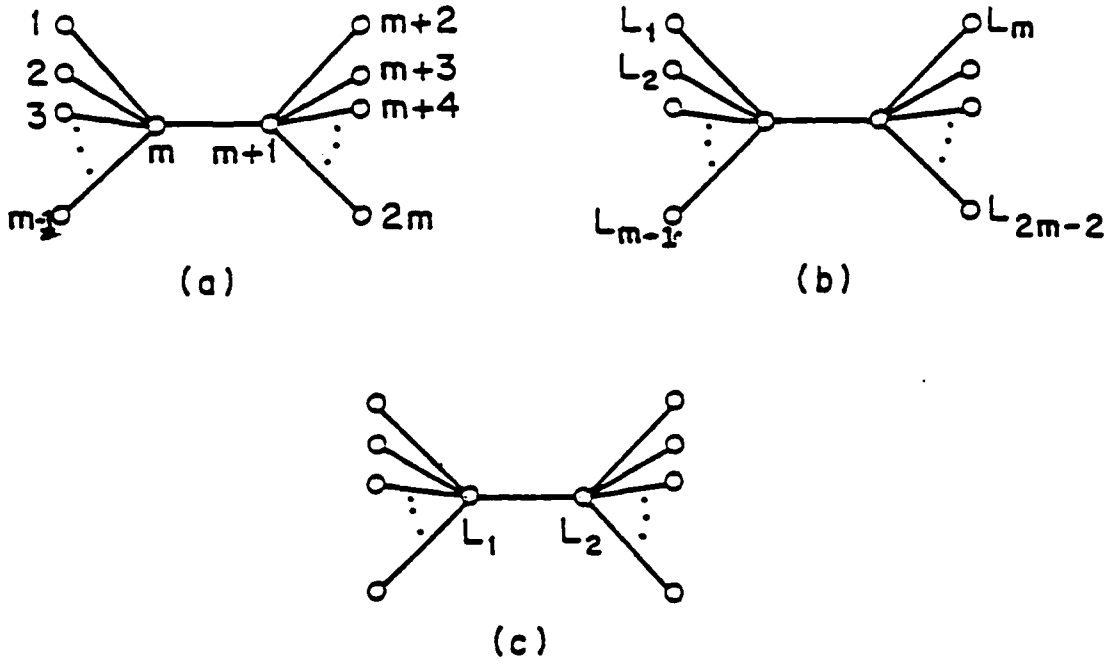


Fig. 1.2.(a), (b) and (c)

These difficulties are explained with reference to the network represented by the graph in Fig. 1.2(a). The nodes represent radios, and the links, bidirectional connections. Notice that transmissions from nodes $1 \ldots m$ conflict with each other, as do those from nodes $m+1 \ldots 2m$. Nodes m and $m+1$ also form a conflicting pair. Any Linked-Cluster organization of this network must have at least 2 clusters since no single node is connected to all the other nodes. figures 1.2(b) and (c) represent 2 possible Linked-Cluster organizations for this network.

First let's focus on Fig. 1.2(b). There are $2m-2$ clusters; the i^{th} cluster is led by L_i. Notice that the i^{th} cluster contains 2 nodes: L_i and either G_1 or G_2. The G nodes are each physically contained in m clusters and also serve as the gateway nodes of the network. Now let us examine the question of which cluster leader(s) should control the gateway node, G_1. Suppose that only one of the cluster leaders, say L_1, controls the transmissions of G_1. Then in a busy-tone solution, G_1 would listen for the busy-tone of L_1 and would transmit whenever the busy-channel is idle. But this could cause collisions in any of the other clusters led by $L_2 \ldots L_{m-1}$, unless different clusters use different frequency bands. The allocation of these frequency bands to the various clusters is not a straightforward problem, and has barely been solved for applications with fixed stations[17]. Since the number of clusters in a stationless environment is constantly changing, an upper bound on the maximum of clusters possible depends on, among other things, the number of nodes in the network. Since any 2 clusters may overlap, the number of bands the access channel is to be divided into, depends in turn on the number of clusters. Hence for large networks the number of frequencies required may become ridiculously large. At any rate, there are a number of reasons why the number of frequency bands the radio can operate at should be kept down to a bare minimum, and so it is highly desirable to avoid the collisions in some *other* way. Notice that similar collisions would occur even if a non-random access scheme such as polling, or reservations were used within each cluster. Thus the organization of fig. 1.2 is not amenable to any efficient solution to the hidden terminal problem. It is important to note that many of these defects can be minimized using spread spectrum, but this introduces the problem of knowing which pseudo-noise keys to listen for, and arguments similar to those made above, now come into play in resolving this problem.

On the other hand if G_1 is controlled by all of the cluster leaders it is connected to, then in a busy-tone solution it can only transmit when it does not sense traffic in any of the clusters $L_1 \ldots L_{m-1}$. This is an unfair throttling of G_1. i.e. the throughput at node G_1 would be very low compared to that of the nodes $L_1 \ldots L_{m-1}$.

Another severe problem in the Linked-Cluster scheme of Fig. 1.2. is that the gateway nodes are common to many pairs of clusters. In fact, no inter-cluster communication can occur without the message being routed through at least one of the gateway nodes. This implies that numerous collisions can occur at G_1 and G_2, since inter-cluster messages are always sent in an uncoordinated manner.

Observe how nicely these defects are corrected in the organization of the nodes in Fig. 1.2(c). Here there are only 2 clusters, and no gateway nodes. The only nodes which are physically contained in more than one cluster are the leaders themselves. This organization is superior to the one in Fig. 1.2(b) for 3 important reasons:

(a) Clusters are larger in size (m versus 2). This decreases collisions since intra-cluster collisions can be minimized using existing techniques.

(b) The protocols for inter-cluster communication are much simpler. This is because the gateway nodes have been eliminated.

(c) The high number of collisions which occured at gateway nodes has been eliminated since there are no gateway nodes.

This example illustrates the point that a Linked-Cluster Algorithm must try to organize the nodes so that there are few clusters, and as few gateway nodes as possible. In this thesis we will examine the important tradeoffs of minimizing the these quantities with the communication and computational complexities of algorithms.

1.5 Previous Work

The only algorithm in the literature which organizes mobile radios into a Linked-Cluster Architecture is due to Baker and Ephremides[5]. It was developed specifically for the Navy's HF Intra Task Force (ITF), which is a general purpose PRN providing extended line of sight communications in the sea. While this distributed algorithm is easy to implement, it has a number of drawbacks which restrict its utility to a very small range of applications. A major aim of this thesis is to suggest ways to overcome these rather serious shortcomings, and so the algorithm (which we will call ITF for a lack of a better name), is now presented and analyzed in some detail.

ITF assumes that the nodes are numbered from 1 to N, where N never changes. The access channel is controlled by Time Division Multiplexing the users. Time is divided into epochs, and each epoch into 2 frames. A frame consists of N slots, and node i broadcasts in the i^{th} slot. The details of what exactly is broadcast in the slots of these frames are not important and can be gotten from [5]. Suffice it to say that by the end of the second frame each node has the following topological information: it knows the nodes to which it is connected, and the nodes to which these nodes are connected. Based on this information a node declares itself to be either a cluster leader, a gateway node, or just an ordinary node. If it is a gateway it knows which clusters it links, and if it is a cluster leader, it knows all its gateway nodes.

The actual algorithm used to form the clusters is a distributed version of the following simple centralized procedure: Declare N to be a cluster leader. Then if $N-1$ is bidirectionally connected to any node including itself that N is not connected to, then declare $N-1$ to be cluster leader. Similarly check if $N-2$ is connected to any node which is not connected to N and $N-1$. If so, declare $N-2$ a leader. The procedure continues like this until all nodes are connected to the set of declared cluster leaders. An example is shown in Fig. 1.3.

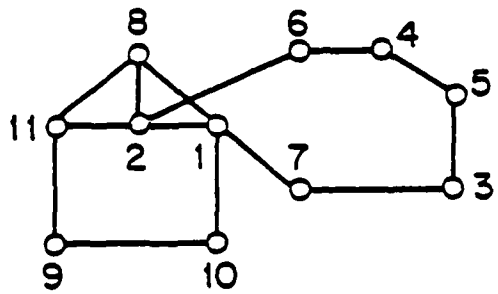


Fig. 1.3
Leaders are selected in the order 11, 10, 7, 6, 5.

Unfortunately, there are a number of difficulties with the algorithm, some of which were recognized and discussed by the authors themselves. We list these previously recognized problems below:

(i) Clusters may be contained within other clusters.

(ii) Several pairs of nodes may declare themselves to be gateway nodes between the same pair of clusters.

(iii) Asymmetric gateway links are possible, in which one of the nodes in a gateway pair is not aware that the other is.

In addition, there remain three major problems which have not been mentioned. The first has to do with the fact that the number of users must be fixed. In any PRN nodes are bound to fail, and other nodes may be needed to be added to the network. The algorithm does not allow for such additions and deletions.

The second problem is that of inefficiency of time. The channel is time division

multiplexed, so that only one user can broadcast at a time. However, it is sufficient to ensure that only users less than three hops away do not broadcast at the same time, to avoid collisions completely. When the network is large and sparsely connected it might be beneficial to risk some collisions, and to then resolve them using a contention based scheme, rather than to TDMA the channel.

The third, and most important problem, from the point of view of this thesis, is that the algorithm does not take into account the fact that it is essential to try to restrict the number of clusters and gateway nodes. In Fig. 1.4. the results of cluster organization are presented using ITF. The reader may verify these results by applying the centralized algorithm illustrated in Fig. 1.3. It is seen that ITF performs miserably sometimes, and is then no better than the case in which every node in the network is a cluster leader, and broadcasts messages at random!

From these remarks one may conclude that the Hidden terminal problem has not been solved for mobile, stationless PRN's, and that much work needs to be done in developing distributed Linked-Cluster algorithms which try to minimize the number of clusters and gateway nodes in the PRN.

1.6 The PRN Model

We now present a model which is reasonable in view of the facts and arguments presented in earlier sections. It is assumed that a PRN consists of N identical mobile radios, distributed in a plane. Radios will be interchangeably referred to nodes and terminals. Each radio has an integer identity number and consists of a receiver, with an omnidirectional antenna, and a transmitter with a fixed transmission radius. All the radios need not have the same radius of transmission. Messages are sent singly by a terminal in framed packets of equal length. The packets contain header information such as the origin and destination, neccessary for multihop routing. If several packets are received at a node simultaneously,

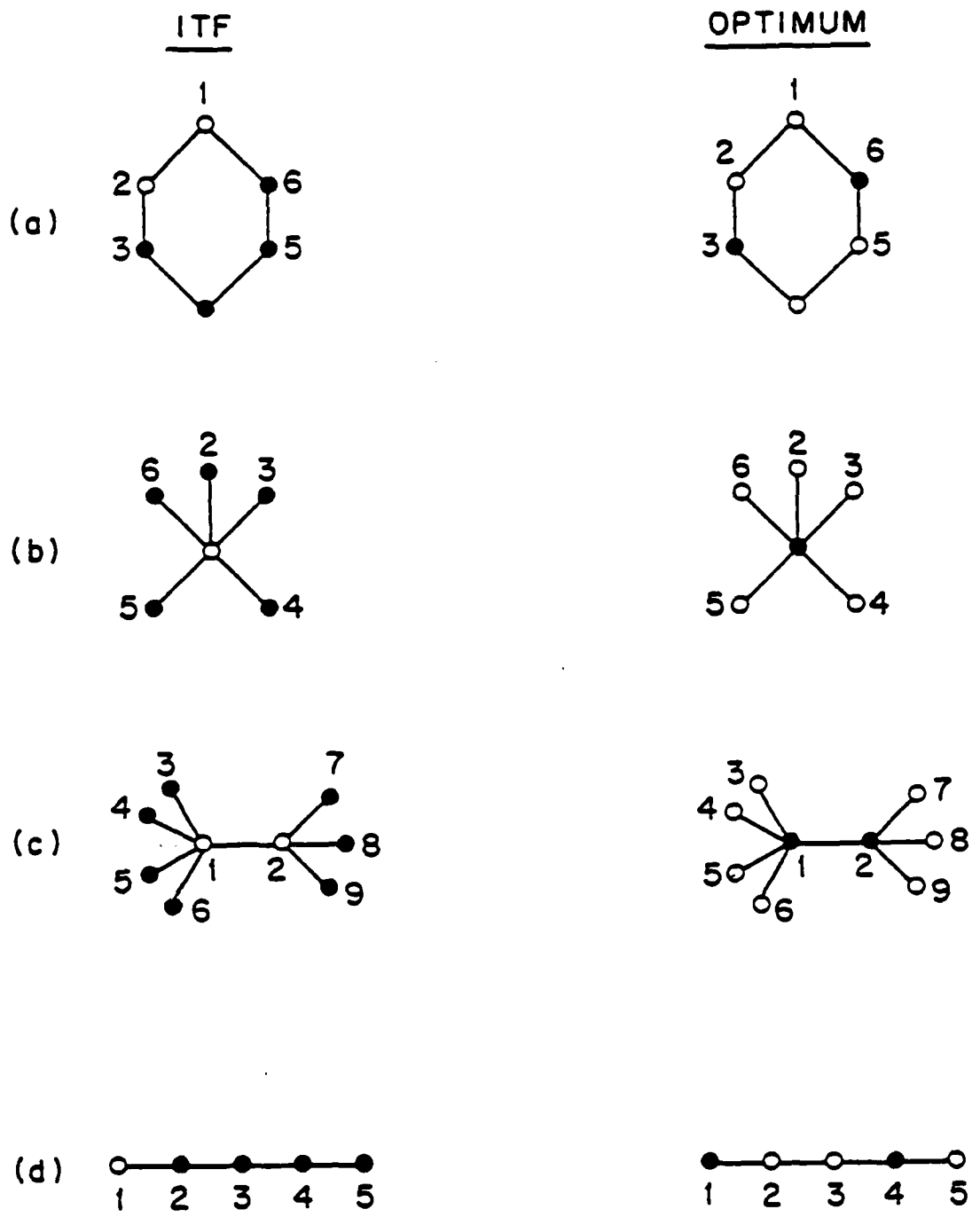


Figure 1.4

all of them are assumed to be destroyed, and must be retransmitted. Lower level protocols such as acknowledgement schemes (ARQ's) which intimate the sender of a lost packet are not assumed to be present. Finally, the PRN is always connected; there is a path between all origin-destination pairs at all times.

1.7. Outline of the Thesis

There are two main parts to this thesis. The first deals with the tradeoff of minimizing the number of clusters and gateway nodes, with the *computational* complexity of the problem. The problem is formulated in graph-theoretic terms, and analyzed from a centralized standpoint (i.e. global information about the topology is assumed.) The resulting optimization problems are found to be NP-complete, forcing us to look for efficient (polynomial time) heuristics. Considerable attention is given to measuring the worst case performances of such heuristics, and it is shown that it is extremely unlikely that an efficient algorithm exists which guarentees a solution that comes to even a constant of the optimum. Then some heuristics are presented and analyzed which are computationally efficient, and which do a "reasonable" job in approximating the optimum answer.

The second part (Chapter 3) consists of distributed versions of the centralized heuristics analyzed in Chapter 2. At this point special emphasis is placed on the communication complexity of the algorithms, and on actual implementation in PRN's.

Finally, Chapter 4 is a brief conclusion, and contains suggestions for further work.

CHAPTER II

COMPUTATIONAL COMPLEXITY OF THE CENTRALIZED PROBLEM

Any Linked-Cluster algorithm must be able to organize the terminals of a PRN efficiently, since the nodes are mobile and the topology of the network may be changing quite rapidly. It has already been argued that the number of clusters and gateway nodes should be minimized. However, the computational implications of such an approach have not been discussed. In this chapter we will address such issues as to the *degree* to which it is possible to minimize these quantities without lapsing into the realm of inefficient (exponential time) algorithms.

In the centralized problem, all the nodes are omniscient and thus have complete information about the topology of the network. Also, computations are performed sequentially, not in parallel. It is easy to see that if the best algorithm which solves the centralized problem runs in time T, then no distributed algorithm can solve the problem in less than time $\frac{T}{N}$, where N is the number of nodes in the network. A direct implication of this is that if no polynomial time centralized algorithm exists, there can be no distributed algorithm which runs in polynomial time either. Hence it is profitable to first study the problem from a centralized standpoint, since negative results apply to the distributed case as well.

2.1. Formulating the Problem

Recall that a cluster leader must be bidirectionally connected to all of the nodes in its cluster. Hence, we will be concerned only with such bidirectional adjacencies in the network. Given a network topology at some time instant, define a graph $G(V, E)$ as follows:

V = the set of users $\{1, 2...N\}$

$E = \{(i, j) : i$ can hear j and j can hear $i\}$.

Observe that a complete set of cluster leaders is a set of nodes such that every node not in the set is adjacent to at least one node in the set. This is known as a *Dominating Set* in Graph Theory, and is formally defined as follows:

Definition 2.1. *A Dominating Set, D, of a graph $G(V, E)$ is a set of nodes such that* $\forall j \in (V - D) \exists i \in D$ *s.t.* $(i, j) \in E$.

If the number of clusters is to be minimized without bothering about the number of gateway nodes, one could look for the minimum cardinality dominating set. If no gateway nodes are desired, then the dominating set should be one of minimum cardinality, such that the graph it induces is connected. A compromise between these two approaches is to find a dominating set of minimum cardinality such that for every pair of nodes in the set there is path which connects them, such that 2 gateway nodes are never in successive order in the path. These three formulations of the problem are now given formally:

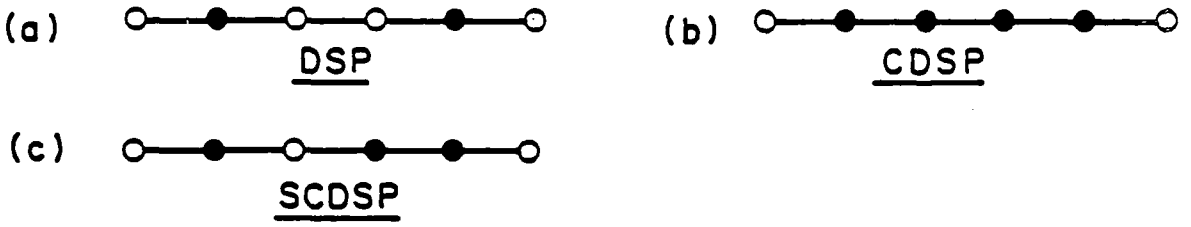
Definition 2.2. *The Minimum Dominating Set problem (DSP) is the one of finding a minimum cardinality Dominating Set, D^*, for a graph G.*

Definition 2.3. *The Connected Dominating Set problem (CDSP) is the one of finding a minimum cardinality Dominating Set, D^c, for a graph G, such that the graph induced by D^c is connected.*

Definition 2.4. *The Semi-Connected Dominating Set problem (SCDSP) is the one of finding a minimum cardinalty Dominating Set, D^s, for a graph, G, such that there is a*

spanning tree of G, whose edges each have the property that at least one endpoint is in D^s.

Fig 2.1 illustrates the different formulations.



	DSP	CDSP	SCDSP
No. gateways	2	0	1
No. clusters	2	4	3

Fig. 2.1

2.2. Some Definitions

In the following sections we will prove a number of results by polynomial many-one reductions of problems widely believed to be intractable to the problems defined above. These intractable problems are defined here:

Definition 2.5. *The Vertex Covering Problem (VC) is the one of finding the minumum cardinality subset of nodes, C, for a graph, $G(V,E)$ s.t. $\forall(i,j) \in E, (i \in C \text{ or } j \in C)$.*

Definition 2.6. *The Set Covering Problem (SC) is defined on a ground set, Γ, a collection, S, of subsets of Γ, and is the one of finding a minimum cardinality subset P, of S such that $\cup_i P_i = \Gamma$.*

Definition 2.7. *The Directed Dominating Set Problem (DDSP) is the one of finding a minimum cardinality dominating set in a directed graph $G(V, A)$.*

Definition 2.8. *The Maximum Leaf Spanning Tree Problem (MLSP), is the one of finding a spanning tree of a graph $G(V, E)$, which contains a minimum number of non-leaf nodes.*

We will also need the concepts of closed and open neighborhoods of a node:

$N(i)$ $i \in V$: the set of nodes adjacent to i.

$\bar{N}(i)$ $i \in V$: $N(i) \cup i$.

$N(S) = \cup_{i \in S} N(i)$.

2.3 NP-Completeness Results

Readers unfamiliar with proofs of NP-Completeness are refered to [13] for a standard treatment of the subject. $A \propto B$ means that any instance of problem A can be converted to an "equivalent" instance of B in time polynomial in the size of the instance of problem A.

Lemma 2.1. *DSP, CDSP, and SCDSP are NP-complete.*

Proof: We prove the lemma in 3 parts:

(a) **DSP**: We show $VC \propto DSP$. Given $G(V, E)$ define $G'(V', E')$ such that

$V' = V \cup \{v_{ij} : (i, j) \in E\}$ and $E' = E \cup \{(i, v_{ij}), (j, v_{ij}) : (i, j) \in E\}$ (see fig. 2.2.)

Suppose that P is a vertex cover in G. Then P is a dominating set in G' (assuming that G has no isolated vertices) since P must be a dominating set in G, and must also cover all nodes of the form v_{ij} in G'.

If D is a dominating set in G', then so is the set

$D^1 = \{v : v \in (V \cap D)\} \cup \{i : v_{ij} \in D\}$. Now observe that since D^1 covers all nodes of the form v_{ij} in G', it must also cover all edges (i, j) in G, thus implying that it is a vertex cover of G.

We can then conclude that G has a VC of size k $\iff$ G' has a DS of size k. Done.

Figure 2.2. illustrates the reduction

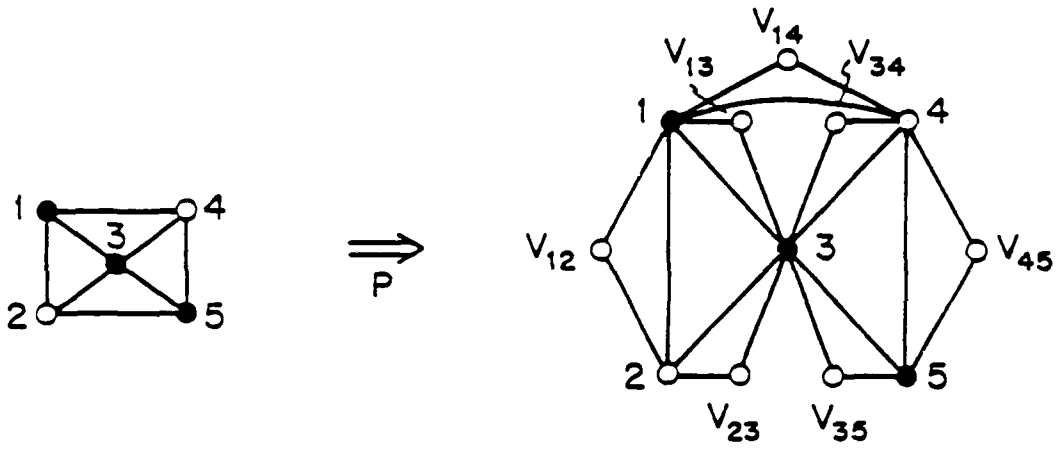


Fig. 2.2

(b) **SCDSP**: $VC \propto SCDSP$. This follows immediately from the reduction in (a). Observe that the dominating set in G' must be semi-connected, else at least one of the v_{ij}'s would not be covered.

(c) **CDSP**: $DSP \propto CDSP$. Given $G(V, E)$ define 2 graphs $G^1(V^1, E^1)$ and $G^2(V^2, E^2)$ such that: G^1 has $|V|$ nodes, $\{c_1 \ldots c_{|V|}\}$, and forms a complete graph; and G^2 has nodes $\{d_1, \ldots, d_{|V|}\}$, and no edges. Then define the graph $G^*(V^*, E^*)$ such that $V^* = V^1 \cup V^2, E^* = E^1 \cup E^2 \cup \{(c_i, d_j) : c_i \in V^1, d_j \in V^2, (i = j)$ or $(i, j) \in E\}$.

See fig. 2.3. for an illustrative example.

Suppose G has a DS of size D. Then the set $\{c_i : i \in D\}$ is a connected DS in G^*. On the other hand, suppose G^* has a DS, $D^* = \hat{C} \cup \hat{D}$ $s.t. \hat{C} \in V_1$, $\hat{D} \in V_2$. Then surely $\hat{D} = \hat{C} \cup \{c_i : d_i \in \hat{D}\}$ is also a CDS, since $N(d_i) \subset N(c_i) \forall i$. Then it is clear that $\hat{D}$ is a DS in G. Done.

All the above reductions can be carried out in time bounded by $O(|V| + |E|)$, i.e they are polynomial time reductions. To see that the problems are in NP observe that

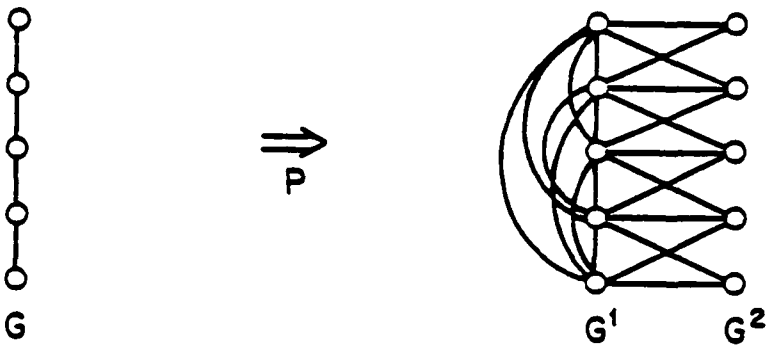


Fig. 2.3

positive instances of the recognition problems can be verified in polynomial time.

Q.E.D.

We note here that while the proofs are original, the fact that both DSP and $CDSP$ are NP-complete is not new [13]. The constructions used in our proofs though are useful later on in the paper.

2.4. Measures of Performance of Heuristics

Since the problem of finding optimal dominating sets is NP-complete we can safely assume that there is no algorithm that solves it in polynomial time. It clear that we are forced to trade optimality with time, but exactly in what proportion is less obvious. Several methods of dealing with NP-complete problems are to be found in the Literature, some more appropriate than others to our problem.

Some algorithms run in polynomial time for most cases, but are inefficient in solving a small number of "atypical" instances. The Simplex algorithm for Linear Programming is such an algorithm. [2] The main reason why such an approach is not appropriate for our problem is that there is a possibility of the algorithm taking an inordinate amount of time in some cases. In a mobile environment, one of most essential attributes of a network organizing algorithm is speed.

Heuristics have also been analyzed with respect to their *average case* performance. This becomes possible when there is reason to believe that the instances encountered in practice obey some probability distribution. In this case *good* algorithms are those which run in polynomial time and are guaranteed to give optimal, or near optimal solutions for "almost" all problem instances.

The notion of "almost all" has been quantified as follows: Let S_N be a given probability distribution for all instances of size N. Let $X(I)$ be a boolean variable for some condition applicable to each of the problem instances. For example, $X(I)$ can be the condition that a given algorithm solves the problem instance I, optimally. Let q_N be the probability that $X(I)$ does not hold for a randomly selected problem instance of size N. Then $X(I)$ holds *almost everywhere* if $\sum_N q_N < \infty$.

[2] It is true that LP is not NP-complete. However, the behaviour of the Simplex algorithm is consistent with the one we are describing.

Now let's examine the suitability of such an approach to our problem. Many researchers have made probabilistic assumptions on the topology of mobile PRN's. For example Nelson has modeled a snapshot of the network as a Poisson point process with a mean density of λ terminals[3]. The radios have identical transmitting radii. Gallager [16] has suggested choosing bidirectional links between terminals with a probability dependent on their distance from each other. Hence, one might try an average case analysis of algorithms based on a reasonable probability distribution for the topology of the network. However, there are 2 problems which remain:

(1) The probabilistic analysis of algorithms is usually extremely cumbersome and involved for the simplest of algorithms, as a consequence of which results have not been forthcoming in this field[13]. There is no reason to suppose that this will not hold for our problem as well.

(2) Suppose that an algorithm is found to be optimal almost everywhere. This does not mean that its performance is guaranteed to be within any reasonable limits for an infinity of "atypical" instances. Suppose the PRN reaches such a configuration, and the radios do not move significant distances for a while i.e. the atypical configuration holds for a long time. Then very poor organization of the network would persist for this period, resulting in numerous collisions, and considerable inconvenience to the users.

In this thesis efficient heuristics will be analyzed from a more conservative standpoint. The performance is based on the *worst-case* situation, thus bounds on the running time and error are guaranteed. The *error* may be measured as a constant difference between the achieved and optimal solutions (*differential error*), or as a constant fraction of the optimal solution (*fractional error*). The next Theorem shows that no efficient heuristic can guarantee a constant differential error for our problem.

Theorem 2.1(a). *Let A be an efficient heuristic for DSP, d_G be the cardinality of an*

solution returned by $\mathcal{A}$ on the graph G, and K_G be the cardinality of an optimal solution of DSP for G. Then unless $P = NP$, there is no integer M such that

$$d_G - K_G \leq M, \forall G.$$

Proof: Suppose there is an efficient heuristic $\mathcal{B}$, for which the theorem does not hold. Then there must be an integer, M', for which the inequality holds. Now consider the graph, H, which is made up $M' + 1$ copies of some arbitrary graph, G, and apply $\mathcal{B}$ to H. By assumption:

$$d_H - K_H \leq M' \tag{2.1}$$

But $K_H = (M' + 1)K_G$. Since the copies of G are disconnected in H, all the nodes in any particular copy, are dominated exclusively by nodes in the same copy. Hence the nodes selected by $\mathcal{B}$ in any copy form a dominating set of G.

Now let the cardinality of the smallest complete set of nodes selected in any copy be γ. It is clear that $\gamma \leq \frac{d_H}{M'+1}$. Substituting expressions for d_H and K_H in 2.1. we have

$$\gamma(M' + 1) - (M' + 1)K_G \leq M'$$

which implies that

$$\gamma - K_G \leq \frac{M'}{M' + 1} < 1.$$

Since the difference on the LHS must be an nonegative integer, we have:

$$\gamma - K_G = 0,$$

which means that $\mathcal{B}$ picks the optimal DS for any G, and that P=NP. Done

Theorem 2.1(b). *Theorem 2.1(a) holds for CDSP and SCDSP.*

Proof: Similar arguments as in part (a).

This negative result leads us to examine the case when error is defined to be a constant *fraction* of the optimum. Efficient algorithms with such error bounds are called ϵ -Polynomial Time Approximate Algorithms, and are defined in the next section.

2.5. ϵ -Polynomial Time Approximate Algorithms

Definition 2.9. *Let A be an optimization problem with positive integral cost function c, and let $\mathcal{A}$ be an algorithm which, given an instance I of A, returns a feasible solution $f_{\mathcal{A}}(I)$; denote the optimal solution of I by $\hat{f}(I)$. Then $\mathcal{A}$ is called an ϵ-approximate algorithm for A for some $\epsilon \geq 0$ iff*

$$\frac{|c(f_{\mathcal{A}}(I) - c(\hat{f}(I))|}{c(\hat{f}(I))} \leq \epsilon$$

for all instances I [13].

The next result, another negative one shows that it is extremely unlikely that an ϵ -PTAA exists for DSP, or for DDSP.

Theorem 2.2. *There exists an ϵ-PTAA for DSP $\iff$ there exists an ϵ-PTAA for SC $\iff$ there exists an ϵ PTAA for $DDSP$.*

Proof: (a) PTAA DSP $\iff$ PTAA SC:

(i) ($\Rightarrow$): Consider an instance of DSP, $G(V, E)$, and suppose there is an ϵ-approximate algorithm to solve SC. Define the following instance of SC:

$\Gamma = V, \quad S = \{N(i) \cup i \text{ s.t. } i \in V\}$

It is clear that the SC instance has the same solution as the DSP instance. So by using the approximate algorithm on the SC problem, we can get an ϵ-approximate solution for the DSP instance.

(ii) ($\Leftarrow$): Consider an instance of $SC, (\Gamma, S)$, and suppose there is an ϵ-approximate algorithm to solve DSP. First define the two graphs G^1 and G^2 where

G^1 is a clique consisting of nodes $V^1 = \{c_1, \ldots c_{|S|}\}$, G^2 consists of nodes $V^2 = \{q_1, \ldots, q_{|\Gamma|}\}$, and no edges.

Next let $G(V, E)$ be such that $V = V^1 \cup V^2, \quad E = E^1 \cup E^2 \cup \{(c_i, q_j) : c_i \in V^1, q_j \in V^2, j \in S_i\}$ (See figure 2.4.)

If P is a solution to the SC problem, then $\{c_i : S_i \in P\}$ is a DS in G. Also, if

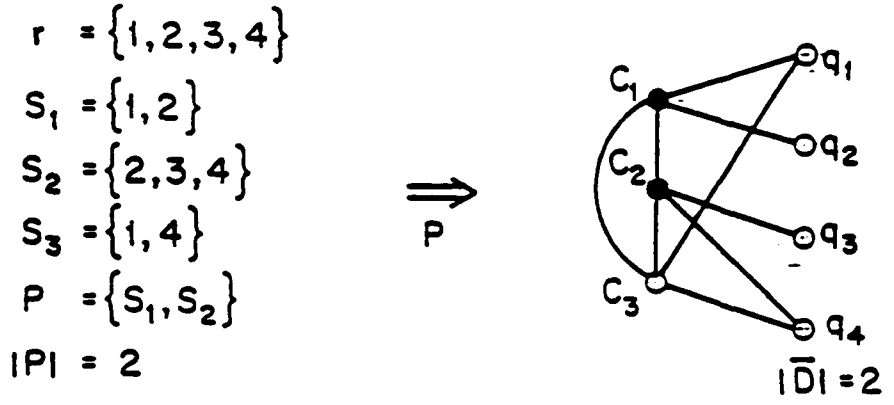


Fig. 2.4

$D = C \cup Q, C \subset V^1, Q \subset V^2$, then so is $\bar{D} = C \cup \{c_i : d_i \in D\}$. But $\bar{D}$ is clearly a set cover for (Γ, S).

We see, that the instance of DSP has the same solution as that of SC, and so using the approximate algorithm on the DSP instance, we get an ϵ-approximate solution for the SC instance. Done.

(b) PTAA *DSP* $\Longleftrightarrow$ PTAA *DDSP*:

(i) (PTAA for *DDSP* $\Rightarrow$ PTAA for *DSP*)

Suppose there is an ϵ -PTAA for DDSP. Then given an instance of *DSP*, G, do the following to the graph:

Replace every edge (i,j), with 2 directed edges (i,j) and (j,i). Now apply the PTTA for *DDSP*, to the modified graph. It is clear that the original graph G, and its modified directed version share exactly the same set of dominating sets, and so the result follows:

(ii) (PTAA for *SC* $\Rightarrow$ PTAA for *DDSP*):

Suppose there is an ϵ-PTAA for *SC*. Then given an instance of *DDSP*, $G(V, A)$ define:

$$\Gamma = V, \text{ and } S = \cup_i^{|V|} S_i \text{ s.t. } S_i = \{j : (i,j) \in A\} \cup \{j\}$$

It is clear that the SC instance has the same solution as the DSP instance. so by using the approximate algorithm on the SC problem, we can get an ϵ-approximate solution for the $DDSP$ instance.

Hence, PTAA $SC \Rightarrow$ PTAA $DDSP \Rightarrow$ PTAA $DSP \Rightarrow$ PTAA SC. Done.

Q.E.D.

Theorem 2.2 is a rather negative result since the Set Covering Problem has been studied extensively [10], [11], [12] and no ϵ-approximate algorithm has been found. In the next lemma, it is shown that $CDSP$ is at least as hard as DSP to find a ϵ-PTAA for, and so we have reasons to be pessimistic for $CDSP$ as well. Interestingly, MLSP is also at least as hard as DSP.

Theorem 2.3. *There exists an ϵ-approximate algorithm for $CDSP \iff$ there exists an ϵ-approximate algorithm for $MLSP \Rightarrow$ there exists an ϵ-approximate algorithm for DSP.*

Proof: (a) PTAA $CDSP \iff$ PTAA $MLSP$

The stronger statement that the 2 problems are equivalent is proved, i.e. $CDSP \iff MLSP$

Let D^c be a solution to $CDSP$ for a graph G. It is claimed that D^c is also a set of non-leaf nodes for a maximum leaf spanning tree of G. Suppose not. Then there is a spanning tree with more leaf nodes. Let the set of non-leaf nodes for this tree be L. Observe that L must form a dominating set in G since each leaf node is connected to at least one non-leaf node. Also, the graph induced by L is connected since the spanning tree is connected. Then, L is a CDS. But L has a smaller cardinality than D^c by assumption, implying that D^c is not a solution of CDSP. The contradiction proves the claim.

Now suppose that T is a solution to $MLSP$, and let L be the set of non-leaf nodes for the solution. We claim that L is also a minimum cardinality CDS. From previous arguments we know that L is a CDS. Suppose it is not of minimum cardinality. Then let

D^c be a CDS with smaller cardinality. We construct a spanning tree as follows: The nodes D^c are connected by any appropriate $|D^c| - 1$ edges, which exist since D^c is a CDS. Now by definition of DS each node not in the DS can be assigned to a single adjacent member of the set. i.e. we can make the nodes not in the DS leaf nodes of the spanning tree. Observe that none of the nodes in D^c can be leaf nodes, since the set is of minimum cardinality. Then by assumption, the constructed spanning tree has a set of non-leaf nodes of smaller cardinality than $|L|$. The contradiction proves the claim.

Figure 2.5 illustrates these arguments.

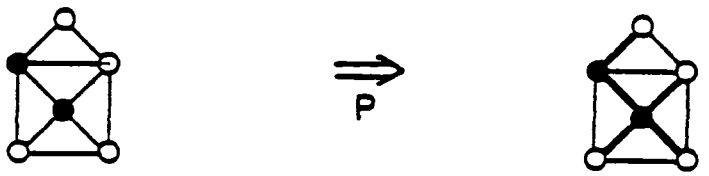


Fig. 2.5

(b) PTAA $CDSP \Rightarrow$ PTAA DSP.

This follows from the contruction in illustrated in Fig. 2.3. Given an instance of DSP, G, use the construction to convert it to the equivalent instance of $CDSP$. Then apply the PTAA for $CDSP$ to the modified graph.

In view of these 2 results, we do not consider it worth pursuing ϵ-approximate algorithms. In the next section we look at algorithms for DSP which are such that the error grows very slowly with the number of nodes in the graph. Unless there is rather large variation in the number of radios in the PRN, such algorithms would minimize the number

of clusters quite efficiently.

2.6. Approximate Algorithms for DSP– The *Greedy* Heuristic

It has been strongly argued in the last few sections that Heuristics for DSP with constant or constant fractional errors do not exist. These negative results suggest that it might be worthwhile to adopt an average-case analysis approach. However, this would not be neccessary if we could show that there are algorithms which have *small* fractional errors which *grow very slowly* with the number of nodes in the network. Such algorithms are in some sense, "almost as good" as ϵ-PTAA's. Fortunately, this is the case for a number of heuristics, some of which are presented here.

We begin by analyzing a very simple *greedy* heuristic. Later on it is shown that some other, more refined algorithms do not do any better, from a worst-case standpoint. This heuristic operates seqentially, putting into the dominating set the least numbered node which covers the maximum number of uncovered nodes in the graph at each iteration. Formally we have:

Heuristic Greedy$(G(V,E))$

Step 0: $UNCOV = V, \quad COV = \phi, \quad i = 1, \quad DS = \phi, \quad C(i) = \bar{N}(i) \forall i \in V.$

Step 1: $d_i = \min arg \max_j |C(j)|, \quad DS = DS \cup d_i, \quad UNCOV = UNCOV - C(d_i)$

$COV = COV \cup C(d_i).$

Step 2: For $i = 1$ to $|V|$,

$$C(i) = \{j : j \in UNCOV and \quad (i,j) \in E, \quad or\, j = i\}$$

Step 3: $i := i + 1$,If COV $\neq V$ then goto Step 1.

Step 4: Stop.

Let m_i be the number of uncovered nodes covered by d_i when it is picked by *Greedy*. And let $|V| = N, \quad K_o =$ cardinality of a minimum DS of G. Since *Greedy* tries to cover

as many uncovered nodes as it can it every iteration, one might wonder what percentage of the nodes it must cover by the time it has picked the optimum number of nodes. In some cases this is 100, but since the algorithm is not optimal, there will be instances for which the first K_o do not do quite as well. The next result provides a bound for this percentage:

Theorem 2.4. *For all graphs G we must have:*

$$\sum_{i=1}^{K_o} m_i \geq N - \frac{N}{e}.$$

Proof: Let A_k be the solution to the following optimization problem, P_k:

$$z_k = max \sum_{i,j \in V} C_{ij} x_{ij}. \tag{2.2}$$

s.t.

$$\sum_{j \in V} x_{ij} = 1 \qquad \forall i \in V. \tag{2.3}$$

$$\sum_{j \in V} y_j = k. \tag{2.4}$$

$$0 \leq x_{ij} \leq y_j \leq 1 \qquad \forall i, j \in V. \tag{2.5}$$

$$x_{ij}, y_j \in INTEGER. \tag{2.6}$$

Define $C_{ij} \begin{cases} 1, & (i,j) \in E, i = j \\ 0, & \text{o.w.} \end{cases}$

This is a formulation of the k-Median Problem with an assignment of edge costs such that it can be applied to the solution of DSP. P_k returns the set of k nodes which covers the most number of nodes in the graph.

Let $u = \{u_1 \ldots u_m\}$ be multipliers for the constraints of the problem P_k. Define:

$$L(x, y, u) = \sum_{i,j \in V} C_{ij} x_{ij} - \sum_{i \in V} u_i \Big(\sum_{j \in V} x_{ij} - 1 \Big) \tag{2.7}$$

$$= \sum_{j \in V} \Big\{ \sum_{i \in V} (C_{ij} - u_i) x_{ij} + \sum_{i \in V} u_i \Big\}$$

So the Lagrangian problem is

$$z_D(u) = \max_{(x,y)} L(x, y, u)$$

$$\text{subject to (2.4), (2.5), (2.6).}$$

Let $z_D = \min_u z_D(u)$ be the Lagrangian Dual. Then we have converted this problem to the form dealt with in [8].

We can now use the following useful result from [8].

$$(z_D - z_g)/(z_D - z_R) < 1/e, \qquad \forall P_k \tag{2.8}$$

where z_R is the minimum objective value of P_k, i.e. $z_R = 0$. z_g is the number of nodes covered in the graph in k iterations of *Greedy*.

Observe that $z_{K_o} = N$, and $z_k < N, \forall k < K_o$, and that $z_k = N, \forall k \geq K_o$. We know that $z_D \leq z_k$. Also, the coefficient matrix of P_k over constraints (2.4) and (2.5) is easily seen to be totally unimodular. This means from Theorem 2 of [9] that z_D is identical to the optimum value of the strong LP relaxation.

From this fact, and from (2.8) we conclude that:

$$(z_{K_o} - \sum_{i=1}^{K_o} m_i)/(z_{K_o} - z_R) = (N - \sum_{i=1}^{K_o} m_i)/(N) \Rightarrow (\sum_{i=1}^{K_o} m_i/N) > (1 - 1/e)$$

and the result follows.

The next Lemma helps to prove one of the important Theorems of this chapter. This theorem gives us the largest possible worst-case fractional error for *Greedy*:

Lemma 2.2. *If Greedy returns a* $DS = \{d_1 \ldots d_{d^*}\}$ *then: (a)* $\sum_{i=1}^{d^*} m_i = N$; *(b)* $m_1 \geq m_2 \geq \ldots m_{d^*} \geq 1$; *(c)* $\sum_{i=1}^{p} m_i + K_o m_{p+1} \geq N \qquad p = 0, 1, \ldots d^* - 1$

Proof: (a) and (b) follow directly from the definition of *Greedy*. At iteration $p+1$ there are exactly $N - \sum_{i=1}^{p} m_i$ uncovered nodes. Let this set be S_{p+1}. Now by definition

of *Greedy* it follows that d_{p+1} covers the maximum number of nodes in S_{p+1}, of all nodes in G. Now consider any optimal DS, DS^*. The average number of nodes in S_{p+1} which are covered by nodes in DS^* is

$$\frac{|S_{p+1}|}{K_o} = \frac{N - \sum_{i=1}^{p} m_i}{K_o}$$

But this average is by definition, at most equal to the number of nodes covered by d_{p+1}, which is m_{p+1}. Then we have:

$$\frac{N - \sum_{i=1}^{p} m_i}{K_o} \leq m_{p+1}, \qquad p = 0, 1, \ldots, d^* - 1.$$

The result follows directly.

Theorem 2.5. *If Greedy returns a DS of cardinality d^* for a graph $G(V, E)$, with optimum $K_o \geq 2$:*

$$\frac{d^*}{K_o} \leq 1 + \frac{1}{K_o} \log_{\frac{K_o}{K_o - 1}} \left(\frac{N}{K_o}\right) \leq 1 + \ln \frac{N}{K_o}$$

and this bound is attained by some graph for all values of $K_o \geq 2$.

Proof: The results of the previous lemma are used to find a bound on d^*. Let T_z be the minimum number of nodes which could ever be covered after $z \leq d^*$ iterations of *Greedy* on a graph G. T_z is obtained by solving the following LP:

$$T_z = \min \sum_{i=1}^{z} m_i$$

s.t.

$$m_1 \geq m_2 \geq \ldots \geq m_z \geq 1, \tag{2.9}$$

$$\sum_{i=1}^{p} m_i + K_o m_{p+1} \geq N \qquad p = 0, 1, \ldots z - 1. \tag{2.10}$$

We know that for the optimal basic feasible solution for T_z exactly z constraints must be satisfied with equality. Now observe that first constraint of the type (2.10) lowerbounds m_1 by $\frac{N}{K_o}$. In general the i^{th} constraint of (2.10) lowerbounds m_i in terms of N, K_o and the values chosen for $m_1, \ldots, m_p$. On the other hand, the constraints of type (2.9) *upperbound* the m_i's. Hence in the optimal basic feasible solution, in which the sum of the m_i's is to

be minimized, as many constraints as possible of the type (2.10) will be satisfied without violating any of type (2.9.) After some algebra we have:

$$m_i = \max\left\{\frac{N}{K_o}(1-\frac{1}{K_o})^{i-1}, \quad 1\right\} \quad i = 1,\ldots,z. \tag{2.11}$$

For $z = d^*$, simplification yields that $m_i = 1, \quad \forall i \geq i_{min}$ where

$$i_{min} = 1 + \log_{\frac{K_o}{K_o-1}} \frac{N}{K_o}. \tag{2.12}$$

Now by summing the geometric series we get,

$$\sum_{j=1}^{i_{min}-1} m_i = N - K_o.$$

But we want to find $\hat{d} : T_{\hat{d}} = N$. So, we have:

$$d^* \leq \hat{d} = K_o + \log_{\frac{K_o}{K_o-1}} \frac{N}{K_o}. \tag{2.13}$$

Dividing through by K_o, the bound follows.

We still have to show that this bound is achieved for all $K_o \geq 2$. This is done by explicitly showing the graph for $K_o = 2$, and then giving a procedure to generalize the construction for arbitrary values. For later reference, we refer to this class of graphs as $B_{k,n}(V,E)$ where $k = K_o$ and $|V| = N = k^{n+1}$.

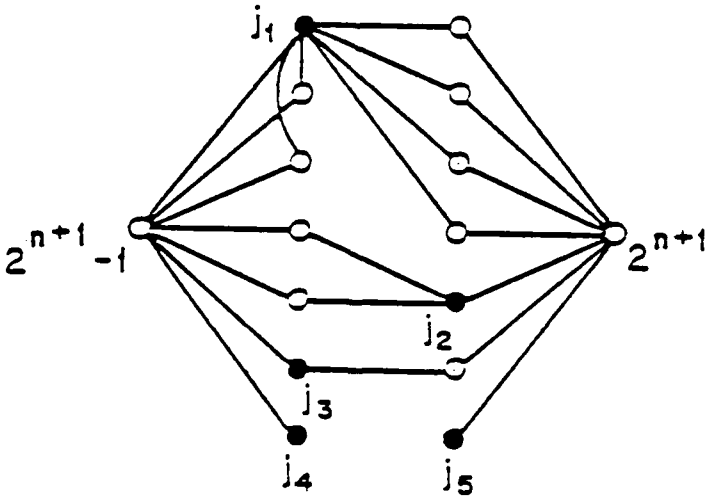


Fig. 2.6 : $B_{2,3}$

Greedy picks $\{j_1, \ldots, j_5\}$ in this example while the optimum is 2. Note from (2.12) that $i_{min} = 4$, and that the conditions of equation 2.11 are met with equality. From this it follows that the bound of Theorem 2.5, which is 2.5 in this case must be attained for the fractional error, as in fact it is. The construction of $B_{2,n}$ is now described: Since there are 2^{n+1} nodes and $K_o = 2$, let

$$N(2^{n+1} - 1) = \{1, 2, \ldots, 2^n - 1\}, \qquad N(2^{n+1}) = \{2^n, \ldots, 2^{n+1} - 2\}.$$

The adjacencies of the j_i's are defined so that *Greedy* picks them in the order of their indices i.e. j_1 is picked first, then j_2 etc. Thus the number of uncovered nodes covered by j_i is simply m_i. Pick $j_1 = 1$ and $N(1) = \{2^{n+1} - 1, 2, \ldots, 2^{n-1} - 1, 2^n \ldots 2^n + 2^{n-1} - 1\}$. It is clear that $|N(j_1)| = 2^n$, and the first constraint of type (2.10) is met with equality j_1 were to be picked first by *Greedy*. Now observe that the least numbered uncovered element of $N(2^{n+1})$ is $3\,2^{n-1}$. Let $j_2 = 3\,2^{n-1}$ and $N(j_2) = \{2^{n-1} \ldots 2^{n-1} + 2^{n-2} - 1, j_2 + 1, \ldots, j_2 + 2^{n-2} - 2, 2^{n+1}\}$ It is easily seen that

$$|N(j_2)| = 2^{n-1} = |N(2^{n+1}) - N(j_1)| = |N(2^{n+1} - 1) - N(j_1)|.$$

Thus we see that j_2 is as attractive a candidate in the second iteration to *Greedy* as are the nodes in the optimal set, after j_1 has been picked in the first iteration. It is possible to give the exact expressions for the remaining j_i's but it is hoped that the construction is clear enough already. The following conditions must hold:

$$|\bar{N}(i) \cap \bar{N}(j_m)| = \left\lceil \frac{N}{2^{m+1}} \right\rceil \quad |\bar{N}(j_m)| = \left\lceil \frac{N}{2^m} \right\rceil, \quad m = 1, \ldots, \hat{d} \quad i = N - 1, N$$

where $\hat{d}$ is defined in (2.13), and $\lceil x \rceil$ is the real number x rounded up. Finally, we must generalize our construction to arbitrary values of k. There are $N = k^{n+1}$ nodes, and the optimal set is $\{N - k + 1, \ldots N\}$. Each of these nodes has a closed neighborhood of size k^n, and thus the neighborhoods are disjoint. Now pick j_1 to be a node in N(N-k+1), and define edges so that it covers exactly k^{n-1} nodes from the neighborhoods of every node in the optimal set. Thus $|N(j_1)| = k^n$. The reader should already begin to see the emerging

pattern since the idea is identical to that in the construction of $B_{2,n}$. Pick j_2 to be a node in $N(N-k+1)-N(j_1)$, and define edges so that it covers exactly k^{n-2} nodes in the neighborhoods of every node in the optimal set, such that these nodes (in $N(j_2)$), are not in $N(j_1)$. The procedure continues analogously for the remaining j_i's. The following conditions must hold:

$$|\bar{N}(i) \cap \bar{N}(j_m)| = \left\lceil \frac{N}{K_o^{m+1}} \right\rceil, \quad |\bar{N}(j_m)| = \left\lceil \frac{N}{K_o^m} \right\rceil, \quad m = 1, \ldots \hat{d}, \; i = N-k+1, \ldots, k.$$

Done.

This result is important for 2 reasons. First it extends work on an equivalent form of *Greedy* which has been analyzed for the Set Covering problem by Johnson, Chvatal, and Hoschbaum [10], [11], [12]. In [11] a best bound on the error is given by $\sum_{i=1}^{\delta} \frac{1}{i}$, where δ is the maximum number of elements in any member of S. Since DSP is a special case of SC, this bound holds for *Greedy(G(V,E))*, but the bound in Theorem 2.5 is tighter. Note that this does not violate Theorem 2.2 since the equivalence holds only for algorithms which have *constant* fractional error.

Secondly, the result tells us that while it is probably not possible to come within a constant of the optimum solution, there are simple algorithms for which the fractional error grows very slowly with an increase in N or K_o. This relationship is plotted in figures 2.7(a) and (b).

Theorem 2.5 gives a bound on the fractional error, but does not provide insight into the actual number of nodes picked by *Greedy* in terms of the size of the graph. We will now show that a famous bound due to Vizing[15], for K_o also holds for d^*:

Theorem 2.6. [3] *For a graph, G, with N nodes and M edges:*

$$d^* \leq N + 1 - \sqrt{2M+1}. \tag{3.1}$$

[3] *The proof of this theorem has been suggested by Prof. Gallager.*

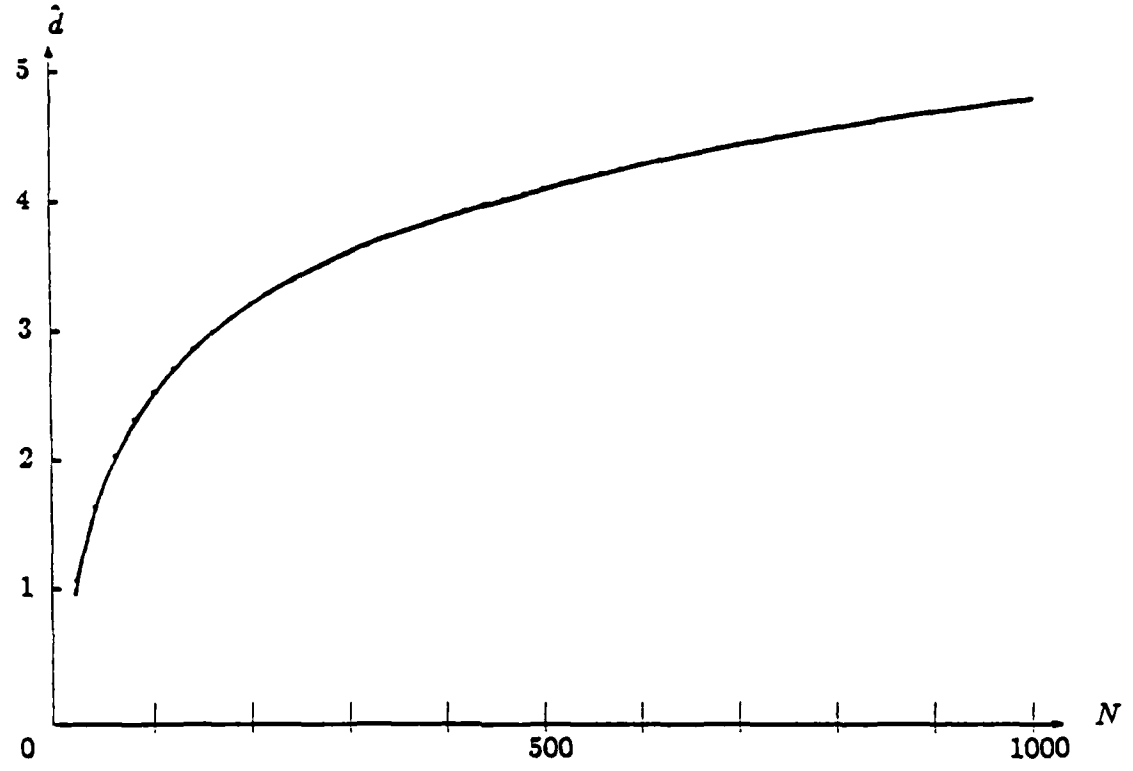


Figure 2-7(a): $\hat{d}$ *vs* N ($K_o = 20$)

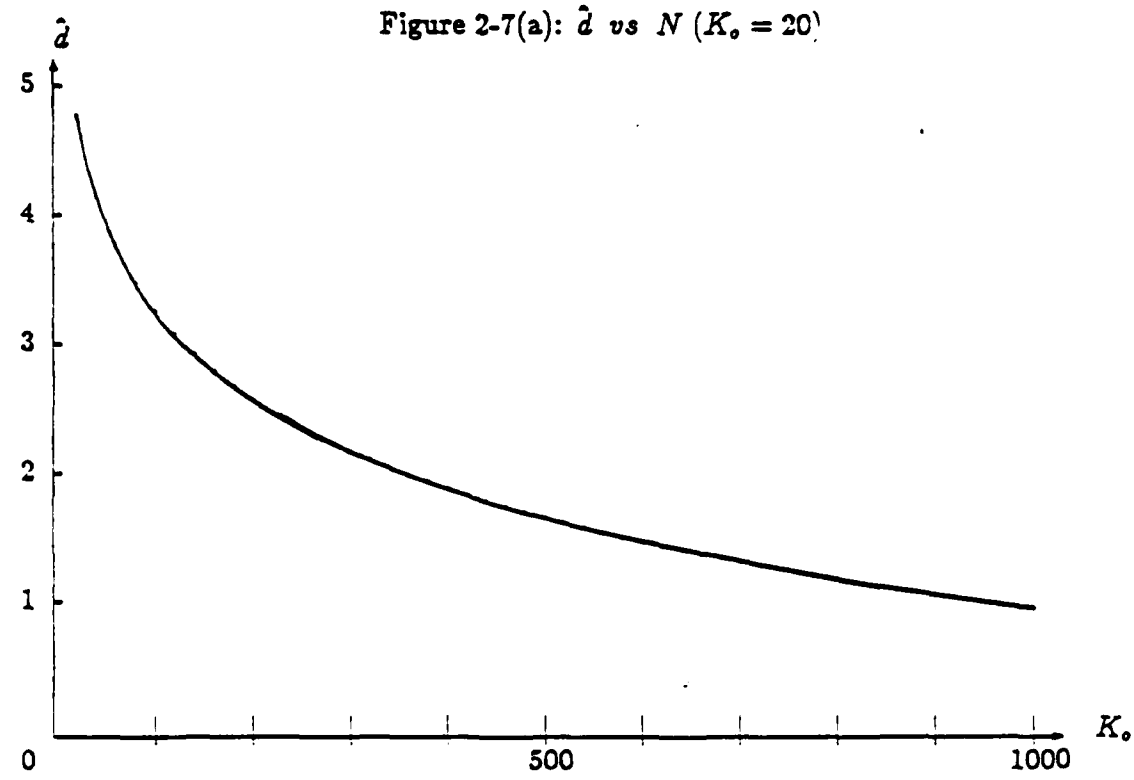


Figure 2-7(b): $\hat{d}$ *vs.* K_o ($N = 1000$)

Proof: First convert G into a directed graph by replacing every edge (i,j) by 2 directed edges (i,j) and (j,i), and by adding self-loops (i,i) at every node i. Interpret a directed edge (i,j) to mean that the free node j would be taken if i were declared cluster head. We can interpret *Greedy* on such a graph as follows: Declare the node with the greatest outdegree a cluster head; delete all edges coming into the neighborhood of that node; and if there are any edges left, declare a new cluster head, else stop. To see that this is identical to *Greedy*, interpret a directed edge (i,j) to mean that if i were picked to be leader, the previously uncovered node, j, would be covered by i. Suppose *Greedy* picks node d_i in the i^{th} iteration. Let $S(i)$ be the set of previously free nodes which were taken by d_i, and let $|S(i)| = m_i$. Finally, define E_i to be the number of edges left at the end of the i^{th} iteration *coming from free* nodes i.e. E_i does not include edges from TAKEN nodes. Set $E_0 = 2M + N$.

Consider some $j \in S(i)$. $|S(j)| \leq m_i$ just before the i^{th} iteration. Since all edges come into FREE nodes, and j is also FREE before the i^{th} iteration, for each edge coming into j, from a FREE node, there is also an edge going out of j to that FREE node. Thus there can be at most m_i edges coming *into* j from FREE nodes, and the total number of edges running from previously TAKEN nodes to members in $S(i)$ is at most m_i^2. There may also be edges from previously TAKEN nodes to members of $S(i)$, but we need not consider them, since we are counting only edges from FREE nodes.

Now observe that we still have to consider the edges coming *from* $S(i)$ into FREE nodes which are not in $S(i)$. These edges are not deleted by *Greedy*, but they are not counted in the definition of E_i. It is clear that the outdegree of every node in S(i) must be $\leq m_{i+1}$, since the self loops of all such nodes has been deleted in the i^{th} step. Thus the number of outgoing edges from $S(i)$, after iteration i is $\leq m_i(m_i - 1)$. This bound can tightened if we know that $d_i \in S(i)$. Then since we know that d_i has zero outdegree, and zero indegree after the i^{th} iteration, we have the bound $(m_i - 1)m_{i+1}$.

By definition of E_i we conclude that:

$$E_i \geq E_{i-1} - m_i^2 - m_i(m_i - 1)$$

$$E_1 \geq E_o - m_1^2 - (m_1 - 1)m_2.$$

$E_{d^*} = 0$ by definition of d^*. Thus,

$$0 = E_{d^*} \geq -\Big(\sum_{i=1}^{d^*} m_i^2 + (m_1 - 1)m_2 + \sum_{i=2}^{d^*-1} m_i(m_i - 1)\Big) + E_0.$$

Solving for E_0 :

$$E_0 \leq \sum_{i=1}^{d^*} m_i^2 + (m_1 - 1)m_2 + \sum_{i=2}^{d^*-1} m_i(m_i - 1). \tag{2.14}$$

Now recall from lemma 2.2. that $\sum m_i = N$ and $m_i \geq m_i - 1 \geq 1$. E_0 can be upperbounded by maximizing the RHS of (2.14) with respect to the m_i's subject to the constraints just mentioned. We claim that this maximum occurs when

$$m_1 = N - d^* + 1, \quad m_2 = m_3 = \ldots m_{d^*} = 1.$$

This is easily seen to be true by contradiction. Suppose the maximum is achieved so that the highest order m_i which is greater than 1 is not m_1, say $m_j, j > i$. Now reduce m_j to 1 and add $m_j + 1$ to m_1. (We can do this because none of the constranints are violated.) Then the difference in the RHS is:

$$\delta = m_j\Big(2m_1 + m_2 - m_j - 1\Big) - \Big(m_1 + m_2\Big)$$

$$= m_j\Big(m_1 + m_2 + m_1 - m_j - 1\Big) - \Big(m_1 + m_2\Big)$$

Now observe that $m_1 \geq m_j \geq 2$.

$$\Rightarrow \delta \geq 2\Big(m_1 + m_2 - 1\Big) - \Big(m_1 + m_2\Big)$$

$$\Rightarrow \delta \geq m_1 + m_2 - 2 > 0,$$

which contradicts the assumption.

Substituting the maximum values in the RHS of (2.14) we have:

$$E_0 = 2M + N \leq (N - d^* + 1)^2 + d^* - 1 + (N - d^*),$$

$$\Rightarrow d^* \leq N + 1 - \sqrt{2M+1}.$$

Q.E.D.

The reader might question here the use of analyzing this centralized, sequential algorithm in such detail, when what is desired is a distributed procedure with a high degree of parallelism. In response we note that a distributed version of *Greedy* is presented in Chapter 3, and claims on its ability to minimize the number of clusters will be based on the analysis just completed.

2.7. Other Approximate Algorithms

In this section it is shown that other reasonable, but more complicated heuristics fail to do better than *Greedy* in the worst case.

From the construction of $B_{k,n}$, one may be tempted to suggest a greedy algorithm which runs *Greedy* N times, each time with a different value of d_1. The output DS would be the minimum cardinality set over all repetitions of *Greedy*. Call this algorithm *Allgreedy* and let the cardinality of its output be D^*. *Allgreedy* performs at least as well as *Greedy*, and also solves $B_{k,n}$ exactly. However, consider the graph $\hat{B}^m_{k,n}$, which consists of m copies of $B_{k,n}$. (Fig. 2.8)

It is easy to see that *Allgreedy* will find the optimum in only one copy of B_k, and will perform in the worst case, as poorly as *Greedy* in the remaining $m-1$ copies. For large m *Allgreedy* will do neglibly better than *Greedy*, in fact the fractional error of *Allgreedy* must approach that of *Greedy* from above, as m increases. Thus even for simple examples *Allgreedy* is nct very effective.

Finally, we examine an algorithm which is not greedy, i.e. nodes selected in one step might be dropped in subsequent ones. The idea behind this algorithm is to maintain

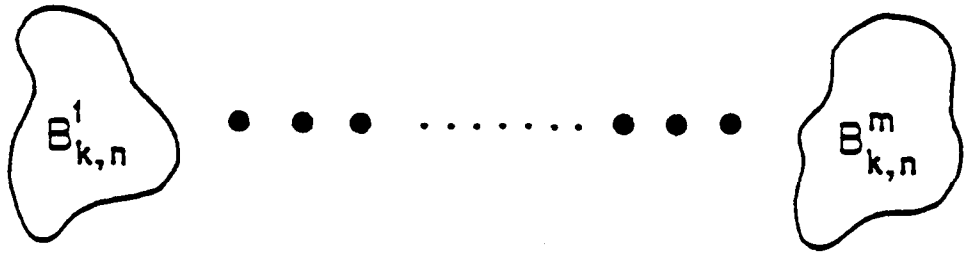


Fig. 2.8

N candidate solutions, such that the i^{th} one includes node i. At each iteration the i^th candidate solution is obtained by choosing i, and the candidate solution from the previous iteration which along with i covers the maximum number of nodes in G. The algorithm is presented below:

Heuristic $Comm(G(V,E))$

Step 0: $S_1(i) = \bar{N}(i), \forall i,\ k = 1,\ D_i = \{i\}$.

Step 1: $k = k+1 \quad j_i^* \in arg\max_{j \in V}\{|\bar{N}(i) \cup S_{k-1}(j)|\} \quad D_i = \{i\} \cup D_{j_i^*}\ \forall i$.

Step 2: $S_k(i) = \bar{N}(i) \cup S_{k-1}(j_i^*) \quad \forall i$.

Step 3: If $\max_{\forall i} |S_k(i)| = |V|$ then STOP, and return the corresponding set D; else go to Step 1.

Initally, the i^{th} candidate solution is just i. In the second iteration (k=2), j_i^* corresponds to the node which covers the most number of uncovered nodes if the only node picked thus far were i. Thus $D_i = \{i, j_i^*\}$ in the second iteration. $S_2(i)$ is nothing but the set of nodes covered by D_i i.e. $N(i) \cup N(j_i^*)$. At iteration k, the algorithm has N candidate solutions, $D_1, \ldots D_N$, each of cardinality at most k. The S_k's represent the nodes covered

by these candidate solutions. The algorithm terminates when one of the solutions covers all the nodes in the graph. The S_{k+1}'s are obtained as follows:
The i^{th} solution set is node i combined with the solution set obtained in the k^{th} iteration, such that the combined set covers the most number of nodes.

Observe that both $B_{k,n}$ and $\hat{B}^m_{k,n}$ are solved exactly by *Comm*, for all k, n, and m. The reader is encouraged to verify this claim. While we cannot show worst case error, we exhibit the following class of graphs R_n, for which the error is potentially arbitrarily large.

$R_n(V_n, E_n)$: Let $|V_n| = 2^n$ and label the nodes $1 \ldots 2^n$. Let i base 2 be the n digit binary representation of $i \leq 2^n$: Now define the adjacencies as follows:

N(1) = { i:i base 2 has a 1 in the first positon }

N(2) = { i: i base 2 has a 1 in the second position }

etc. until

N(n-2) = { i: i base 2 has a 1 in the $n - 2^{nd}$ position }

N(n-1) = { i: i base 2 has 01 in the last 2 positions }

N(n) = { i: i base 2 has 10 in the last 2 postions }

N(n+1) = { i: i base 2 has 00 in the last 2 positions }

N(n+2) = { i: i base 2 has 11 in the last 2 positions }

Observe that the optimal $DS = \{n-1, n, n+1, n+2\}$. However, *Comm* picks the set $\{1, 2, \ldots, n+2\}$. To see why this happens, first observe that we need to focus attention only on the nodes $1, \ldots, n+2$ since these nodes have much larger neighborhoods than any of the other nodes for moderate values of n. In particular, nodes $1, \ldots, n-2$ have degree 2^{n-1} and nodes $n-1, \ldots, n+2$ have degree 2^{n-2}. It is convenient to call $A = \{1, 2, \ldots n-2\}$ and $A' = \{n-1, n-2, n, n+1\}$. Now observe that:

$$|N(K) \cup N(i)| = |N(K)| + 2^{n-2} - 0.25|N(K)| = 0.75|N(K)| + 2^{n-2}, \quad K \subset A \cup A', i \in A' - K \tag{2.15}$$

$$|N(K) \cup N(j)| = |N(K)| + 2^{n-1} - 0.50|N(K)| = 0.50|N(K)| + 2^{n-1}, \quad K \subset A \cup A', i \in A' - K \tag{2.16}$$

We see that

$$0.75|N(K)| + 2^{n-2} < 0.50|N(K)| + 2^{n-1} \Rightarrow |N(k)| < 2^n. \tag{2.17}$$

From (2.17) it follows that for the first $n-2$ iterations all of the candidate solutions have either no elements which belong to A', or at most one element. In iteration $n-1$, $D_i = A \cup \{j\}$, $j \in A'$, $\forall i$. The remaining three iterations proceed by picking 3 nodes in A' so that by the end of the $n + 2^{th}$ iteration, $D_i = A \cup A' \quad \forall i$. *Comm* is off by exactly $n-2$, implying that the error can become arbitrarily large, and that it varies roughly as ln N. Thus *Comm* is not a favorable alternative to *Greedy*.

Observe that the fractional errors of *Allgreedy* and *Comm* are the same for the class of graphs, R_n.

2.8. Summary

In this chapter the complexities of minimizing the number of clusters and gateway nodes in a PRN were examined from a centralized standpoint. It is extremely unlikely that an algorithm exists which could come to even a constant of the optimal solution and still run in polynomial time. However, there are algorithms for DSP, with errors that grow extremely slowly with the number of nodes. We have given best possible bounds for the performance of some such heuristics, and doubt seriously the existence of a heuristic which performs significantly better in the worst case.

No results have been obtained for heuristics for $CDSP$, and $SCDSP$. However, minimizing the number of clusters certainly has a diminishing effect on the number of nodes in the backbone network. This cuts down, to some extent, the amount of communication necessary to deliver inter-cluster messages. In the next chapter, emphasis will be on minimizing the communication complexity and the number of gateway nodes, within the framework of distributed algorithms, based on heuristics whose computational complexity

was analyzed in this chapter.

CHAPTER III

LINKED–CLUSTER ALGORITHMS

The results obtained thus far have concentrated on the centralized problem, and it now time to relate them to the issue of devising efficient distributed algorithms which minimize the number of clusters in mobile, stationless PRN's.

A major part of this chapter deals with a linked cluster algorithm called *GCLA*, which minimizes the number of clusters identically to *Greedy*. First, collisions and acknowledgement problems are ignored to make the mechanics of the algorithm and its equivalence to *Greedy* easier to understand. This version is called *DISTG* for the Distributed Greedy Algorithm. Subsequently strategies for collision minimization and resolution are presented. Another algorithm called *Tree Linked Cluster Algorithm* is also included. *TLCA* does not minimize the number of clusters as well as *GCLA*, but it has a significantly lower overhead of communication.

3.1. The Distributed Greedy Algorithm (*DISTG*) Framework

In this section the model for subsequent analysis is presented and its assumptions justified.

(a) The network itself is represented as a finite directed graph, $H(V, A)$ with N nodes and P edges. V is the set of users, and an edge is defined for a node pair (i,j) if and only if j can hear i. However, in most analyses only bidirectional connections will be considered for reasons to be explained later. In this case we obtain an unidirected

graph $G(V, E)$, where

$$E = \{(i,j) : (i,j), (j,i) \in A\}$$

Let there be M undirected edges in G. It is assumed that G is connected.

(b) Nodes must send messages on all the directed (and therefore undirected) edges emanating from them. This is to capture the broadcast nature of the network.

(c) Errors in transmission are not allowed i.e. there are underlying acknowledgement schemes. Other lower level protocols such as error detection and message formatting are also assumed.

(d) Messages arriving at a node are queued, and then processed on a first come first serve basis.

(e) The delay across any edge is finite but variable. Messages are assumed to be received along a direction of an edge in the order that they were transmitted.

(f) Interference is ignored and is to be dealt with later.

This model is a variation of the one presented in [14] which deals strictly with point to point networks. Also, observe that (a) is a "snapshot" representation of the mobile network.

3.2. Performance Measures for Distributed Algorithms

Two measures of performance have traditionally been considered for distributed algorithms, both predicated on the notion of an *elemental message*. An elemental message is an integer representing a node identity, the cardinality of a set, or some other such quantity. The first measure of performance is the *Communication Complexity*, C, and is an asymptotic bound on the number of messages *sent* in the worst case, as a function of the

problem size. In a network the problem size is usually the number of nodes, or the number of nodes plus the number of edges M.

The *Time Complexity* T, is a measure of the total amount of time it takes to run the algorithm in the worst case. Most of this time is taken by communication among nodes, since typically, many computation steps can be carried out in the time that it takes a message to be transmitted and received. So unless the algorithm is highly inefficient computationally, it is reasonable to assume that computation time is negligible. Observe, that if the algorithm takes exponential time, the assumption of zero computation time falls through. Hence, Time complexity is measured as the asymptotic bound on the number of units of time required, if the communication of an elemental message accross an edge takes one unit of time.

It is important to realize that these measures do have their limitations. Firstly, they are asymptotic, and do not account for possibly large multiplicative and additive constant factors. Since most mobile PRN's are not very large, asymptotic results can at best be viewed as crude measures to compare different algorithms for the same problem. For this reason, the *actual number* of elemental message units will be presented with any results for C or T. Secondly, the results are worst-case, and therefore not necessarily indicative of typical performance. Lastly, for most packet networks there is a large overhead in sending many short messages, as against sending a few longer ones. This is primarily due to the fact that much header information goes into every packet, and our definition of elemental message does not include this. To partially counter this effect, we will also give the total number of *broadcasts required*, with any results for C.

Since there are two performance criteria for distributed algorithms something must be said about the tradeoffs between them. In commercial wide area point-point networks, C has an impact on the availability of the network for other functions, whereas T has an impact on the grade of service offered to the users. In the tactical environment, these impacts can be

viewed as the level of security (i.e. likelihood of enemy interception) versus the timeliness of decision making. In most networks, the amount of communication required is more sensitive to fluctuations in dollar terms, than is the time taken to run the algorithm, and so C is generally considered to be more crucial. Also, C always upperbounds T. However, in some PRN's the value of timely service may be much more than that of less communication, and so there is no application independent manner by which the tradeoffs of these measures can be examined.

3.3. The Distributed Greedy (DISTG) Algorithm

The Distributed Greedy Algorithm finds a Dominating Set for the underlying graph G, without taking into account the effects of interference. Recall that G consists of undirected edges which represent the bidirectional connectivities between node-pairs. The reason for not including the other edges is that we would like eventually to extend this algorithm to a truly broadcast algorithm called the Greedy Linked-Cluster Algorithm ($GLCA$), in which it is assumed that no acknowledgement schemes are present as lower level protocols. If the connection is not bidirectional, then messages cannot be acknowledged in a direct manner, and the problem becomes extremely complicated. $DISTG$ contains virtually all of the attributes which will enable us to claim that $GLCA$, the extended algorithm, minimizes clusters as well as *Greedy* does. Also, the correctness of $GLCA$ follows straightforwardly from that of $DISTG$.

$DISTG$ is entirely event-driven in its communication–there are no time outs. This is desirable since it eliminates the need for accurate timers, but more importantly because it limits the amount of communication. The algorithm is conducive to a high degree of parallelism– different parts of the network may be in entirely different stages, and the nodes terminate based on events which have occured, not on some period of time. This clearly plays a role in keeping T down as well. A convenient assumption is that every node

knows the (bidirectional) connectivities of itself, and those of its neighbors at the start of the algorithm. This can be achieved easily by a 2 stage flooding scheme.[5]

The idea of the algorithm is the following: Suppose a node, i, covers the largest number of uncovered nodes in its second neighborhood. This means that if at that point in time, any of i's neighbors, say j, is to be covered, then i would be that node in the graph which would cover the largest number of nodes and which also covers j. So it is reasonable to assert that a node, with only very local knowledge of the network topology would like to elect as cluster leader its neighbor which covers the largest number of uncovered nodes. If all nodes can be dominated by their most "highly connected" neighbors, a reasonably small number of clusters can be expected. *DISTG* fulfils this goal, as we will see.

Data Structures at node i:

Connectivity: a list of edges which describe $N(\bar{N}(i))$.

Status: a variable which takes values from {FREE, HEAD, TAKEN}, depending on whether i is uncovered, a leader, or covered but not a leader.

Nstatus: a list indexed such that Nstatus(j) is the status of $j \in \bar{N}(i)$.

Uplist: a list of tuples such that $(a, b) \in$ Uplist iff necessary changes in information on node b which are caused by the declaration of a as a leader, have been made at node i.

UpdateStatus(a): a boolean variable which is either WAIT or NOWAIT. It is WAIT if there is at least one node in N(i) from which new information is likely to be received because of the declaration of node a as a cluster head.

k^*: the lowest indexed node of $\bar{N}(i)$ which has a maximum cardinality set of FREE neighbors. This is clearly the most highly connected neighbor of the node as discussed above.

Kstatus: a list of the k^* values of all the node's FREE neighbors.

Flag: a variable which is set either UP or DOWN. It is UP iff UpdateStatus(a)=WAIT for some node a, or if no information has been received from the start of the algorithm on k^* for some neighbor of i. One can view Flag as a necessary check on extreme degrees of parallelism among neighboring nodes, which could result in outdated events being acted

upon. When Flag is set to UP, one must interpret this as a step in the algorithm when the node wants to update its information on the k^* values of its neighbors.

3Sent: a list of boolean variables, each set to TRUE or FALSE. 3Sent(l)=TRUE means that i is 3 hops away from a cluster head, l, and that it has broadcast the necessary information pertaining to this fact.

Leader: the identity of the cluster leader of i.

The algorithm is now presented. We focus on how it works at a node, i. The initialization is as follows:

Flag=UP since the node has no information on the k^* values of its neighbors.

3Sent(l)=FALSE $\forall l$;

Leader= i;

UpdateStatus(a)=NOWAIT for all $a \in N(\bar{N}(i))$;

Nstatus(j)=FREE $\forall j \in \bar{N}(i)$;

Uplist, Kstatus=ϕ;

Send(k_i^*);

As the values of k_i^* are received from neighbors, Kstatus is updated. Once values of k^* have been received from all neighbors, the procedure *TryHead* listed below, is executed. At this point i checks to see if it is the choice of all of its neighbors. If so, it declares itself a cluster leader, changes its status to HEAD, and terminates the algorithm after broadcasting the change of status to its neighbors. It also broadcasts N(i); we will see why this is done later. *TryHead* is executed, as shown later, in response to various messages.

Event TryHead:

begin

If (Flag=DOWN) **AND** ($Kstatus(j) = i \ \forall j \in$Kstatus) **then** (*is i ripe? *)

begin

```
            Leader=i;
            Nstatus(i) := HEAD;
            Send(i is a head,N(i));                    [ type0 message]
            TERMINATE;
        end
end
```

Let us now consider the response of i to the message that one of its neighbors, j, is a head. First it records the information that j is a head in Nstatus. A_j (see code) is the set of nodes in N(i) which are affected by j's declaration. These sets enable Nstatus, Kstatus, and Uplist to be modified to record all changes in N(i) due to j's declaration. After this, i must pass on the new state to its neighbors if in fact there are neighbors which do not know that j is a cluster head, and which have not terminated. This is necessary because:

(a) If i has changed its status, its neighbors must know that if they were to declare themselves cluster heads, they would cover at least one less node. Also, they must now disregard the value of k_i^*.

(b) Suppose $k_p^* = i$ for some neighbor p. It is clear that this value can change, once the number of FREE nodes covered by i does. Assume that the value changes to q, where q is a node which would declare itself to be cluster leader, if only its free neighbor p, would change its value of k^* to q. Then we see that in order for q to be able to change its status, i must communicate the new status of the node i, along with the information for p to calculate the size of i's new FREE neighborhood.

Hence, i broadcasts the fact that it is TAKEN by j, and N(j). Note that if there are no FREE nodes in $N(i)$, i will broadcast the information and then terminate. This situation is illustrated by figure 3.1, and the code presented below:

Event DoTaken: executed when *Rec*(Type0 from j):

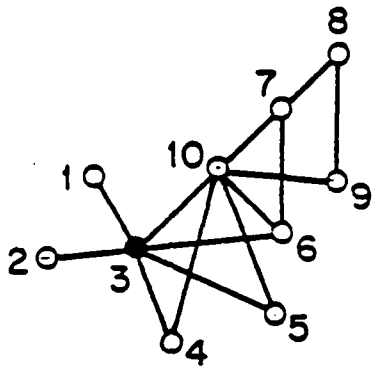


Fig. 3.1.
After 3 declares itself a HEAD, nodes 7 and 9 will prefer 8 instead of 10.

```
begin
        If (j,i) ∉ Uplist then
        begin
                If Leader=i then Leader=j;      (*assign i to leader j*)
                (* Find i's affected neighbors *)
                A_j = N̂(i) ∩ {k : k ∈ N(j) and Nstatus(k)≠HEAD};
                Nstatus(j) = HEAD;
                Delete Kstatus(j);      (*j is not FREE anymore *)
                For all p ∈ N(j) s.t. Nstatus(p)≠ HEAD do
                        Nstatus(p)=TAKEN;
                (* Modify the datastructures *)
                For all k ∈ A_j do
                begin
                        Delete Kstatus(k);
                        Uplist=Uplist∪{(j,k)};      (*Since i ∈ A_j, (l,i) ∈Uplist *)
                end;
                (* Any nodes in N(i) which do not know that j is a HEAD are told so. *)
```

```
If ∃k ∈ N(i) − N̄(j) such that (j,k)∉ Uplist then
begin
        UpdateStatus(j)=WAIT;
        Send(i taken by j, N(j));    [type1 message]
end
else
begin
        UpdateStatus(j)=NOWAIT;
        If Kstatus(m) = i for some m ∈ Kstatus then
                TryHead;        (*Since Flag may be down *)
end;
If {k : k ∈ N̄(i), Nstatus(k)=FREE} = φ then TERMINATE;
    end
end
```

Observe that if i broadcasts, it also sets UpdateStatus(j)=WAIT. The reasons for doing this are best illustrated by the simple example of 5 nodes connected in a line. (Fig. 3.2.) Another thing to note is that i does not send a value of k_i^* since it is now TAKEN.

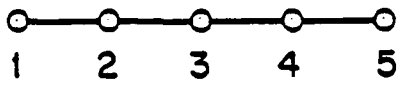


Fig. 3.2

It can easily be seen that the only node that can declare itself to be a HEAD, after Flag=DOWN for the first time, is 2. This node sends out the information that it is a leader to nodes 1 and 3. Node 3 receives this, and in executing Event *DoTaken*, sends out its broadcast. Now suppose UpdateStatus(j) is not set to WAIT. Then it is possible for Flag to be DOWN. Observe that the most current value of k_4^* 3 possesses is 3. Its only other neighbor is not free, and so event *Tryhead* has occured. Hence 3 becomes a cluster head. Similarly, 4 and 5 will also become cluster leaders. By ensuring that Flag=UP, we force 3 to wait until node 4 has had a chance to reevaluate its value of k^*, which is clearly 4 itself.

We now move to the case when i receives a message sent by a node j, whose neighbor l, has just declared itself a cluster leader. If i is just one hop away from l then it will execute *DoTaken* subsequently, if it has not done so already. Otherwise i must be 2 hops away from l. C_l is the set of nodes in N(i) which are affected by l's declaration. When i receives a type1 message pertaining to a leader l for the first time, it has enough information to modify all its variables correctly: All i's non-head neighbors, adjacent to l are in the set C_l which is determined as shown in the code. Hence (l,i) is added to Uplist, and all subsequent type1 messages pertaining to l are ignored.

Again, i sets UpdateStatus=WAIT, but only if there is a chance of it declaring itself a cluster leader incorrectly. Fig. 3.3. shows this condition, and also illustrates some of the points made earlier:

Event TwoAway: executed when *Rec*(Type1 message from j; l is leader)

begin

(* Continue only if i is 2 hops away, and has not heard about j's declaration *)

If $(l,i) \notin$Uplist and $l \notin N(i)$ then

begin

Uplist=Uplist$\cup$\{ (l,j)\};

$C_l = \{N(i) \cap N(l) \cap \{k$:Nstatus(k)$\neq$HEAD$\}\}$;

Nstatus(l)=HEAD;

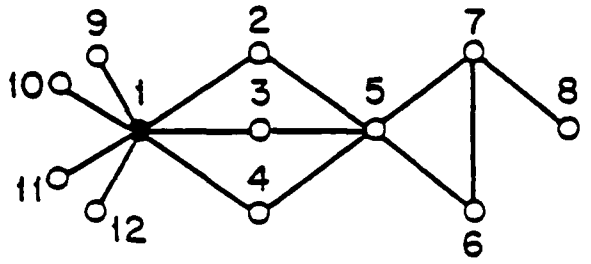


Fig. 3.3

(*Modify datastructures*)

For all $k \in C_l$ do

begin

Delete Kstatus(k);

Uplist =Uplist$\cup\{(l,k)\}$;

end;

For all $p \in N(l) \cap N(\tilde{N}(i))$ s.t. Nstatus$\neq$HEAD do

Nstatus(p)=TAKEN;

If Nstatus(i)=FREE then Recalculate k_i^*;

Send(l,N(l), and [k_i^* if Nstatus(i)=FREE]); [type2 message]

Uplist=Uplist$\cup\{(l,i)\}$;

If $\exists p : (l,p) \notin$ Uplist, Kstatus(p)=i then

UpdateStatus(l)=WAIT

else

begin

UpdateStatus(l)=NOWAIT;

TryHead;

end

If $\{k : k \in \bar{N}(i)$, Nstatus(k)=FREE$\} = \phi$ **then** TERMINATE;

end

end

Node 1 has declared itself a leader, and node 5 has received a message from, say node 3, intimating it of this fact. Now 5 has only 2 FREE neighbors left (6 and 7), both of which had initially chosen 5 to be their most highly connected neighbor. If UpdateStatus(l) were not set to WAIT, then it is possible that Flag=DOWN, and 5 would declare itself to be a leader, which it clearly should not be able to do. When Flag=UP, 5 must wait to be informed that 7 is now the most attractive candidate.

However, if i is none of its FREE neighbors' most highly connected neighbor, it is not necessary to change UpdateStatus(l) to WAIT. If i is FREE, the new value of k_i^* must be broadcast. This is calculated easily from Connectivity and Nstatus.

We now consider i's reponse to a type2 message. There are 3 possibilities: i is either 1, 2, or 3 hops away. In the first and second cases, no communication is necessary, but i must update its variables. This is done as shown in the code. It is possible that UpdateStatus=NOWAIT, and i may execute *TryHead* subsequently.

If i is three hops away from l it must send an updated value of k_i^*. Again, i has all the information necessary to do this the very first time it receives a type2 message pertaining to l: E_l is the set of neighbors of i which are two hops away from l, and not HEADs. At this point i can find its most highly connected neighbor from Connectivity and Nstatus. Hence, k_i^* can be broadcast without waiting to hear from the other nodes, if in fact Nstatus(i)=FREE. To prevent i from rebroadcasting another type3 message pertaining to l, the boolean 3Sent is set to TRUE.

Observe that while i cannot calculate the new values of k^* of all members of E_l, it need not wait to hear from all of them before executing TryHead. This is because all of i's

neighbors that preferred it before l's declaration will continue to do so after the declaration. (See fig.3.4. for an illustration of this point.)

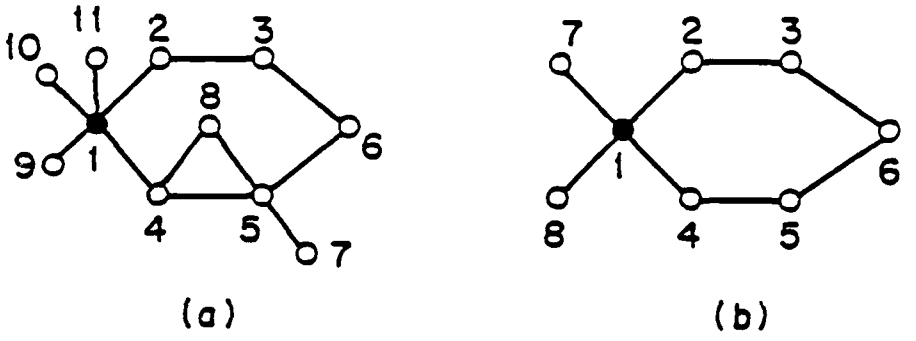


Fig. 3.4.

Node 1 declares itself a HEAD. Suppose node 6 receives a type2 message from node 3, but has not heard from node 5. In 3.4 (a), Kstatus(5) $\neq$ 6 at node 6, and so 6 will not declare itself head before hearing from node 5. In 3.4(b) Kstatus(6)=5 before 6 has heard from 5, and this value will not change even after it has.

Event OneTwoThreeAway: executed when *Rec*(Type2 from j; l is leader)

begin

If ($l \in N(i)$) **OR** ($l \in N(N(i))$) **then** (*i.e. if i is 1 or 2 hops away from l *)

begin

If Nstatus(j)=FREE **then**

Replace the value of k_j^* in Kstatus;

Uplist=Uplist $\cup\{(l,j)\}$;

If $\nexists k \in N(i) - \{l\}$ s.t. (l,k)$\notin$ Uplist **then**

begin

UpdateStatus(l)=NOWAIT;

TryHead;

```
            end;
        end
        else begin      (*i is 3 hops away *)
            E_l = {k : k ∈ N(i), Nstatus(i)≠HEAD s.t. ∃p ∈ {N(k) ∩ N(l)}
                                                AND Nstatus(p)≠ HEAD};
            If Nstatus(i)=FREE then Recalculate k_i^*;
            For all p ∈ N(l) ∩ N(N(i)) s.t. Nstatus≠HEAD do
                Nstatus(p)=TAKEN;
            If not 3Sent(l) then      (*Only one broadcast per head *)
            begin
                Send(l, and [k_i^* if Nstatus(i)=FREE)];  [type3 message]
                3Sent(l)=TRUE
            end;
            TryHead;
        end
end.
```

The only remaining case is the action taken on receiving a type3 message: When this happens i checks if it is 2 hops away from l. If it is not, then it must be 3 or 4 hops away and merely updates its Kstatus. If it is 2 hops away, it replaces the new value of k^*, and modifies Uplist appropriately. It then checks to see if the conditions have been met to CLOSE UpdateStatus(l).

```
Event TwoFourReceive: executed when Rec(Type3 from j; l is leader);
begin
        If l ∈ N(N(i)) then
        begin
            Uplist=Uplist ∪{(l, j)};
            Replace k_j^* in Kstatus;
```

If $\nexists k \in N(i)$ such that $(l,k) \notin$ Uplist then

begin

UpdateStatus(l)=NOWAIT;

TryHead;

end

else

UpdateStatus(l)=WAIT;

end

else

Replace k_j^* in Kstatus; TryHead;

end.

3.4. Correctness of *DISTG*

The approach taken is to show that *DISTG*, and a centralized algorithm *Multigreedy* produce exactly the same dominating sets for all graphs. *Multigreedy* is sequential, but may pick more than one node for the dominating set in a single iteration.

Multigreedy converts the undirected graph G to a directed one. This is done by replacing every edge in G by 2 directed edges going in opposite directions. In addition a loop is added at each node. Interpret a directed edge, (i,j), to mean that if i were to be chosen at that stage of the algorithm, then j would be covered. Initially, all nodes are uncovered so if a node is chosen then its entire closed neighborhood is covered. This explains step 0 (see code). Following, is some notation that helps explain operations on the directed graph:

$N^+(i)$ = the outneighborhood of node i.

A Ripe node is defined as follows: i is Ripe $\Longleftrightarrow$

$$\Big[|N^+(i)| > |N^+(j)| \, OR \quad |N^+(i)| = |N^+(j)| \, AND \quad i < j\Big] \quad (\forall j \in (N^+(i)).$$

Some explanation is in order. i is ripe if and only if it is the most "highly connected node"

of all its FREE neighbors. The free neighbors of i are $N^+(i)$. None of the neighbors of these free neighbors of i should have a greater outdegree than i. This gives the expression.

Algorithm Multigreedy: Input: An undirected graph, G(V,E).

(0) Replace every edge (i,j) in E by 2 directed edges (i,j) and (j,i). For every node i, add (i,i). Set $D = \phi$.

(1) $K = \{i : i \text{ is Ripe}\}$.

(2) $D = D \cup K$

(3) Delete all edges coming into $\bar{N}(K)$.

(4) If any edges remain, goto step 1; else **STOP**.

Output: a dominating set, D.

Once a node i is chosen, by our interpretation of directed edge, all edges *incoming* to the nodes in $\bar{N}(i)$ must be deleted. This is done in step 3. If there are any edges left at the end of an iteration, there are still some uncovered nodes remaining, and so step 1 is repeated.

Observe that the k^* values in *DISTG* help determine which nodes are Ripe in the network a particular stage of the algorithm. The deletion of an edge in *Multigreedy* is much like the book-keeping done for Nstatus in *DISTG*. In the following argument we will verify that only Ripe nodes are declared cluster leaders by Procedure *TryHead* in *DISTG*.

After the initial values of k^* have been received at the least numbered node with the highest outdegree, and its Flag=DOWN for the first time, the node will declare itself to be a cluster head, broadcast, and terminate. This ensures that the algorithm will in fact

begin. Note that this node is Ripe in the first iteration of *Multigreedy*.

The next Lemma will be useful in the proof.

Lemma 3.1. *If 2 nodes declare themselves to be cluster leaders at the same time, they must have no FREE neighbors in common.*

Proof: By contradiction. Let nodes p and q declare themselves to be leaders simultaneously, and let their neighborhoods have some FREE node, r in common. Since both nodes consider themselves to be ripe, r must have considered each one of them its most highly connected neighbor at different times. Suppose, without loss of generality that q was an earlier value than was p. When k_r^* changed to p, it must have done so because of a decrease in $|N^+(q)|$. It is now argued that r could not have learned of this decrease from q itself.

Suppose q knew that its FREE neighborhood had decreased, before its declaration. When q broadcast this type1 or type2 message, it had to have set UpdateStatus(t)=WAIT for some t, implying that Flag must have been UP. Thus it must have received the modified value of k_r^* before setting UpdateStatus(z)=NOWAIT, i.e. making it possible for Flag to be DOWN, and so it could not consider itself to be Ripe. This contradiction shows that q could not have known of the decrease in its FREE neighborhood before declaring itself a cluster head, implying that r could not have heard of the decrease from q.

So suppose r heard from some $u \neq q$, and let t be the node whose declaration resulted in the decrease in $|N^+(q)|$. It is clear that q and t must have at least one node, g in the intersection of their closed neighborhoods which was FREE before t's (and q's) declaration. Suppose $g = q$. Then $k_q^* = t$, and q could not have declared itself a HEAD without hearing about t's declaration. If $g = t$ then Kstatus(t)=q at q, implying that t could not have declared itself HEAD. So g is some node other than q and t. Now since t's declaration preceeds q's, it follows that when g was FREE, $k_g^* = t$. Then Nstatus(g)=t at q when it declares itself HEAD because q does not know of t's declaration. This contradiction

shows that g does not exist, implying that the FREE neighborhood of q does not change by t's declaration $\Rightarrow$ k_r^* does not change $\Rightarrow$ p could not have declared itself a cluster head until it learned that q had.

Q.E.D.

Now suppose that a node i declares itself a cluster leader at some time, T of the algorithm. There are 2 possible cases:
Node i was a FREE node just before it became a HEAD.
Node i was a TAKEN node just before it became a HEAD.

Case 1 : In general some of i's neighbors are free, and others already taken when i becomes a leader. Note that none of the neighbors can be leaders, since i is FREE. Now, suppose that $N(i)$ consists of *no* TAKEN nodes. Then the value of $|N^+(i)|$ has not changed since the beginning of the algorithm. Since, i is going to declare itself a HEAD, all of its neighbors must have regarded it as their most highly connected neighbor at some earlier time in the algorithm. Now observe that values $|N^+(i)|$ never increase, implying that since none of the nodes in $|N(i)|$ has been TAKEN, i still has to be the most highly connected node of all its neighbors.

But suppose that some of i's neighbors are TAKEN nodes. Let j be the neighbor to be most recently taken before time T. Note that this time must be strictly before T, because no 2 nodes can simultaneously become cluster leaders if they have a Free node in common. The Free node in this case is i. Now suppose that some FREE neighbor of i, say p does not have $k^* = i$ at time T. Obviously, i does not know this since it is about to declare itself a HEAD. This means that i was p's most highly connected neighbor at some earlier time. This situation could only have changed by a subsequent *decrease* in the value of $|N^+(i)|$. The last time this happened was when j was taken. At this time, i must have sent a type 2 message to node p, and set its Flag to UP. At time T, i's flag is DOWN, which means that node p considered i to be its most highly connected neighbor even after $|N^+(i)|$

had reduced to its value at time T. This contradicts our assumption, and *DISTG* always picks a Ripe node in this case.

Case2 : Since i is taken at time T, at least one of its neighbors has already declared itself a cluster head. Let j be the neighbor to become a head most recently before T. Before i received the type0 message from j it could not have considered itself Ripe because j, its neighbor did, and we know that 2 neighbors can never consider themselves Ripe simultaneously. So after receiving the type0 message from j, i set its Flag to UP and broadcast a type1 message to all its neighbors. All its free neighbors must have declared it to be the most highly connected node, and therefore we know that even after the most recent reduction in $|N^+(i)|$, due to a type0 message, node i is the most ripe nodes. However, observe that $|N^+(i)|$ could also have been reduced because of type1 messages received. That this cannot lead to an error has already been shown in Case 1. From this we conclude that if i is TAKEN at time T, it cannot declare itself to be a leader unless it is ripe.

We still have to show that the algorithm never terminates before covering all the nodes, and that it never deadlocks. The second of these issues is resolved easily in light of the preceding discussion. *DISTG* can never deadlock because there are always Ripe nodes in the graph, until all nodes are covered. This follows from the fact that *Multigreedy* does not deadlock. Now we show that the algorithm will never terminate if there is even a single uncovered node in the network. This can be seen from the stopping conditions at a node– either the node is a cluster leader, or it has no FREE neighbors. An uncovered node always has status FREE, and so none of such a node's neighbors can terminate until it has been covered.

This completes the proof of correctness for *DISTG*.

3.5. Equivalence of DISTG and *Greedy*

We have shown *DISTG* and *Multigreedy* pick identical Dominating sets; hence it is sufficient to show that *Multigreedy* and *Greedy* behave identically, in order to establish the equivalence of *Greedy* and *DISTG*. For any graph G, with some order of numbering on its nodes, both *Greedy* and *Multigreedy* have unique solutions. Let R_G, and T_G respectively, be these solutions for an undirected graph $G(V, E)$.

Theorem 3.2. $\forall G, T_G = R_G$.

Proof: Let $R_G = \{g_1, g_2, \ldots g_x\}$, where g_k is the node picked in the k^{th} iteration. Define S_i to be the set of uncovered nodes covered by g_i when it is picked. As before let $|S_i| = m_i$.

Now let $T_k = \{t_1 \ldots t_x\}$ be the set of nodes picked in the k^{th} iteration of Multigreedy. Observe that 2 nodes cannot be picked in the same iteration of *Multigreedy* if they have any FREE nodes in the intersection of their neighborhoods. Define C_i to be the set of uncovered nodes covered by t_i when it is picked. Clearly the C_i's are mutually disjoint Also, let $|C_i| = b_i$. By the definition of the algorithms, we know that $|\bigcup_{i=1}^{k} T_i| \geq k$, which implies that it is sufficient to show that $\bigcup_{i=1}^{k} T_i \subset R_G \forall k$. We show the following by induction on k, the number of iterations:

(*i*) $\cup_{i=1}^{k} T_i \subset R_G$

(*ii*) $\{g_1, \ldots, g_k\} \subset \cup_{i=1}^{k} T_i$

(*iii*) $g_i = t_j \Rightarrow S_i = C_j$.

(a) *Basis* $k = 1$: Initially, all the nodes are uncovered. *Multigreedy* selects least numbered nodes which have the maximum sized neighborhoods in their own second neighborhoods. Trivially, g_1 has this property, i.e. $g_1 \in T_1$. Now suppose that $i \in T_1$ for some $i \neq g_1$. At some iteration of *Greedy* at least one of the nodes in

$\bar{N}(i)$ will be covered for the first time. Let j be such a node. By definition, i is the least numbered node in the graph which covers the largest number of nodes including j, so it follows that i will be picked by *Greedy*. Let $i = g_w$. Now observe that the set of uncovered nodes covered by i is the same when it is picked by either algorithm. i.e. $C_i = S_w = \bar{N}(i)$.

(b) *Inductive Step*: $k = K + 1$: By assumption Multigreedy has picked $g_1, \ldots, g_k$, and possibly some more nodes in R_G. Also if $g_i = t_j \Rightarrow S_i = C_j$ for all t_j's picked in previous iterations. Now let's look at g_{K+1}. If it has already been picked in previous iterations of *Multigreedy*, then condition (ii) of the hypothesis is met. Suppose g_{K+1} was not picked in previous iterations. By the hypothesis we know that all the nodes in S_{k+1} must be uncovered in the $K+1^{st}$ iteration of *Multigreedy*. Then since $\{g_1, \ldots, g_K\} \subset \bigcup_{i=1}^{K} T_i$, it follows that g_{K+1} is Ripe, and therfore that *Multigreedy* picks it in iteration $K + 1$. Now suppose that $i \neq g_{K+1}$ is Ripe in the $K + 1^{st}$ iteration. Consider the set $|N^+(i)|$. None of these nodes is covered in the first k iterations of *Greedy*. Thus the first node, j, in the set is covered in *Greedy* after iteration K, say iteration w. The node which covers it in *Greedy*, g_w, must be Ripe at this stage. Since j is the first member of $N^+(i)$ to be covered, it follows that i must be the highest connected neighbor of j i.e. $g_w = i$. Hence we have shown that all three conditions of the hypothesis hold.

Q.E.D.∎

We have already shown that *DISTG* returns the same dominating set as *Multigreedy*. This, with the above result, establishes that *DISTG*, and *Greedy* minimize the number of clusters identically. Based on the work in Chapter 2, it can thus be strongly argued that *DISTG* minimizes clusters as efficiently as could be hoped for.

3.6. Complexity of *DISTG*

Initially, the list *Connectivity* must be established at each node. This takes $2N$ broadcasts. All subsequent communication is triggered by the declaration of a leader. The number of broadcasts resulting from node i becoming a leader is no more than $F_i = |N(N(N(i)))|$. So, the total number of broadcasts $\leq \sum_{i=1}^{d^*} F_{d_i} + 2N$

Now observe that the number of elemental messages in each of type0, type1 and type2 messages sent is bounded by the size of the neighborhood of some cluster leader, plus 2. Type3 messages consist of only 2 elemental messages. Let δ be the size of the largest neighborhood. Also, let $T_i = |N(N(i))|$. We have:

$$C \leq \sum_{i=1}^{d^*} T_{d_i}(2 + |N(d_i)|) + (F_{d_i} - T_{d_i})2 + 2N\delta \tag{3.2}$$

(Recall that d^* is the number of nodes picked by *Greedy*. To obtain an accurate expression for C which depends only on M and N, we will make the assumption that the network is regular i.e. all nodes have the same degree. Then $|N(d_i)| = \frac{2M}{N} = \delta, \forall i$. It is also assumed that $\delta >> 2$, and that communication for establishing Connectivity is negligible. So that (3.2) is simplified to

$$C \leq d^*(T_{d_i}\delta + (F_{d_i} - T_{d_i})).$$

The analysis of C is broken into 3 cases, depending on the density of the graph. This is because (3.1) gives a tight bound for dense graphs, whereas the actual number of messages exchanged pertaining to a particular cluster head varies directly with the density. We will also make use of the bound for d^* derived in Theorem 2.6.

Case 1: $\delta^3 \leq N$: It is easily seen that $T_{d_i} = 1 + \delta^2$ and $F_{d_i} - T_{d_i} = \delta(\delta - 1)^2$. Thus

$$C \leq d^*((1 + \delta^2)\delta + \delta(\delta - 1)^2).$$

$$\Rightarrow C = O(\delta^3 d^*) \leq O(N^2 - N\sqrt{1 + 2M} + N) = O(N^2 - N\sqrt{1 + 2M}) \tag{3.3}$$

Case 2: $\delta^3 > N \qquad \delta^2 \leq N$: As in Case 1,

$$C \leq d^*((1 + \delta^2)\delta + \delta(\delta - 1)^2).$$

$$\Rightarrow C = O(\delta^3 d^*) \leq O(\delta N d^*) \leq O(2MN - 2M\sqrt{1+2M} + 2M) = O(MN - M\sqrt{1+2M}) \tag{3.4}$$

Case 3: $\delta^2 > N$: Then

$$C \leq N\delta d^* \leq 2MN - 2M\sqrt{2M+1} + 1 = O(MN - M\sqrt{2M+1}) \tag{3.5}$$

To summarize:

$$C = \begin{cases} O(N^2 - N\sqrt{2M+1}), & 2^{\frac{3}{4}} M^{\frac{3}{4}} \leq N \leq M+1 \\ O(MN - M\sqrt{1+2M}), & \frac{1+\sqrt{8M+1}}{2} \leq N < 2^{\frac{3}{4}} M^{\frac{3}{4}} \end{cases}$$

Observe that the performance degrades with the density of the graph. It is important to note that it may be possible to modify the algorithm to eliminate all broadcasts of neighborhood sets of leaders. This would surely complicate the protocol, and it is for reasons of exposition that we chose not to do this. However, such a modification would reduce C by a factor of about $\frac{2M}{N}$, resulting in better performances (about $O(N)$ and $O(N^2)$) for the 2 cases in the summary above.

The Time complexity is determined quite simply. Every time a leader is declared, 4 types of messages are broadcast, and all of the type are broadcast at the same time. So

$$T \leq 4d^* = 4(N - \sqrt{2M+1} + 1) \Rightarrow T = O(N - \sqrt{2M+1}). \tag{3.7}$$

3.7. The Greedy Linked Cluster Algorithm (GLCA)

The aim of this section is to extend *DISTG* to accomodate effects such as interference, and the lack of lower level acknowledgement schemes.

There are 2 kinds of collisions: inter and intra leader. Recall that all communication in *DISTG* is triggered by a node declaring itself a leader, and consists of 4 kinds of messages. By intra-leader collisions we refer to those collisions which occur because of the

communication triggered by the same leader. Inter-cluster collisons are defined analogously. Since a node has rather restricted knowledge of what is going on in the rest of the network, collisions are inevitable within the framework of $DISTG$.

Type1 intra-leader collisions can be eliminated quite simply as follows: include a schedule of how the the neighbors of a newly declared leader should broadcast within the type0 message. This is done without any wastage of the access channel because the leader knows the connectivities of its neighbors, and can thus tell when 2 or more of these neighbors can broadcast simultaneously. When a type1, or type2 message is broadcast by a node, it can similarly schedule the broadcasts of its neighbors, thus minimizing collisions to some extent. However, in these cases the method may not be efficient.

Inter-leader collisions appear to be very hard to predict, and there may be no way to minimize them. Notice however, that Type0 inter-leader collisions never occur since 2 nodes may declare themselves to be leader only if they do not share any FREE neighbors.

The next step is to describe a broadcast protocol which resolves the collisions which do occur. Consider the following simple scheme. Time is slotted such that the length of a slot is equal to the time taken to broadcast a packet. There is a global clock so that the slots are perfectly synchronized. Packets are broadcast at the beginning of a slot. The transmission probability, p_t^i of a node, i, is the likelihood of it transmitting a packet in a given slot given a message to be transmitted. We set $p_t^i = (\max_{j \in \tilde{N}(i)} |\tilde{N}(j)|)^{-1}$. This access scheme has a throughput of at least $\frac{1}{e}$. To see this suppose that node i has A neighbors. It is clear that $p_t^j \leq \frac{1}{A+1}\ \forall j \in \tilde{N}(i)$. The conditional probability of a packet from i getting through in some slot, given that i does transmit a packet in that slot is thus $\geq (1 - \frac{1}{A+1})^A$, which converges from above to $\frac{1}{e}$. Retransmission probabilities for collided packets are adjusted to maintain the values of p_t^i. Unfortunately, expected delays might be quite high for this scheme, but it seems quite difficult to develop multiaccess strategies for this situation which assure both high throughput and low delay. Note that since the

packet lengths are very small, a TDMA scheme over all nodes of the network might be reasonable in terms of throughput and delay. However, every node would then have to know the identities of all the other nodes in the network; also, additions and deletions of nodes would become difficult.

We now deal with the problem of acknowledgements. It is clear that the existence of some acknowledgement scheme is essential to the working of any protocol. Observe that type1 messages acknowledge type0 messages, type2 messages acknowledge type1 messages, and type3 acknowledge type2. However, type 3 messages are never acknowledged, and other messages need not be acknowledged by all the sender's neighbors. Another unacknowledged message is the initial broadcast of k^*. It is clear that while the algorithm does provide some form of acknowledgement for many of the messages, it does not deal with the problem adequately. The precise nature of a more complete scheme depends on the kind of application, in particular on the nature of the broadcast channel. For a good explanation of the issues involved see [18].

3.8. The Tree Linked Cluster Algorithm (TLCA)

While *DISTG* minimizes the number of clusters very efficiently, it does this at a high cost of communication, and is only applicable to sparse graphs. If the PRN is extremely dynamic and the control algorithm must be run very frequently, then *DISTG* is not a good choice. When a low value of C is crucial, our standards of optimality should be lowered. In presenting *TCLA*, we will first use the *DISTG* framework (from section 3.1), and then suggest ways to deal with collisions later.

The idea behind *TLCA* is as follows: Initially every node knows its second neighborhood, calculates its value of k^* (the identity of the least numbered neighbor with maximum degree), broadcasts this value, and waits to hear the corresponding values from all its neighbors. So far the algorithm is identical to *GLCA*. However, the values of k^* are not allowed

to change for the rest of the algorithm. Now observe that some nodes will not be preferred by any of their neighbors. One such node is the highest numbered node of smallest degree. We call such nodes *leaves*. A leaf forces its preferred neighbor to become a cluster head and then terminates. This new head then broadcasts its status, and all its non-terminated members broadcast the fact that they are now covered. When a non-head recieves a message that one of its neighbors is covered, it disregards that neighbor's value of k^* and then checks to see if it is a leaf. Thus all communication is triggered by Leaf nodes. The code of the algorithm is presented below:

Data Structures at node i:

Connectivity: a list which contains the adjacencies of the first and second neighborhoods of i.

Status: a variable which takes values from $\{TAKEN, FREE, HEAD\}$.

k_i^*: a variable with value equal to the identity of the least numbered neighbor of i with the maximum degree in N(i).

Lstatus: a list of neighbors $j : k_j^* = i$. i.e. those neighbors which prefer i.

Leader: the identity of the cluster leader of node i.

Initialization at node i::

Status=FREE;

Leader=i;

Send(k_i^*);

Rec(k_j^*) from all neighbors;

Update Lstatus;

Event CheckLeaf: executed after receiving k^* from all neighbors.

```
begin
        If Lstatus=φ then
        begin
                Send(k_i^* become a HEAD);
                Leader=k_i^*;
                Status=TAKEN;
                TERMINATE;
        end
end.

Event BeHead: executed when Rec(i become a HEAD);
begin
        Leader=i;
        Status=HEAD;
        Send(i HEAD);
        TERMINATE;
end.

Event TellCovered: executed when Rec(j HEAD);
begin
        If Status=FREE then
        begin
                Status=TAKEN;
                Leader=j;
                Send(i covered);
        end.
```

end.

Event Revise: executed when Rec(j covered);

begin

Lstatus=Lstatus-$\{j\}$;

CheckLeaf;

end

A few comments are in order. Whenever a node is covered by a head it sets Check=FALSE, because it need never force another node to become a HEAD. However, if a FREE node recieves a message that one of its neighbors has been covered, it must then check to see if it is now a leaf. The algorithm terminates at a node when it becomes either a HEAD or a leaf. If a node is neither, it follows that $Lstatus \neq \phi$, and there is some uncovered neighbor which prefers it. Therefore the algorithm should not terminate in this case. When all nodes are covered it is clear that the algorithm will stop at all nodes.

The complexity of *TLCA* is now analyzed. A node does not broadcast more than 4 times. Two broadcasts go in establishing the list *Connectivity*. Subsequently a node may be a leaf, in which case it broadcasts once; or a HEAD in which case it broadcasts once to declare its status, and it may have had to broadcast once earlier to indicate that it was covered by a neighbor. Thus the number messages broadcast is:

$$C \leq 4M \Rightarrow C = O(M)$$

The Time complexity depends on the number of clusters. This number is trivially bounded by N. For every cluster leader declared, one broadcast comes from the leaf; one from the head, and one from each of the non-terminated neighbors of the head. Thus

$$T \leq 3N \Rightarrow T = O(N).$$

This represents a substantial improvement in complexity over *DISTG*. However, some accuracy is lost in minimizing the number of clusters, and this loss is reflected in a lower value of T. For example, look at Fig. 3.5.

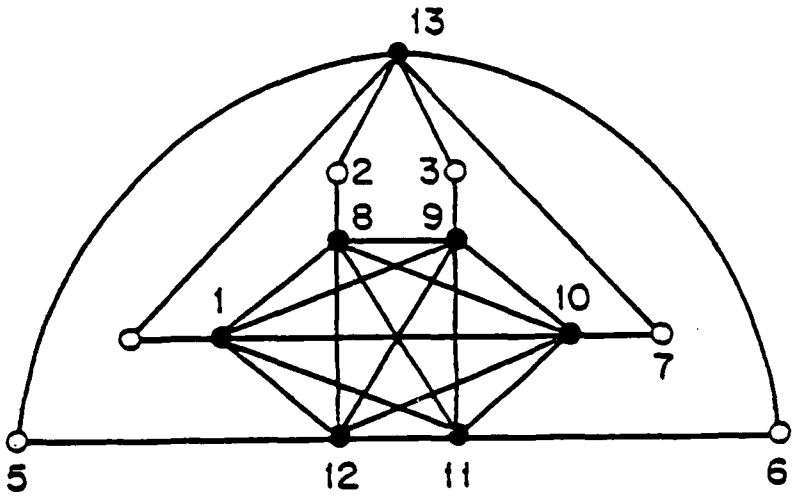


Fig. 3.5

TCLA picks $\{1, 8, 9, 10, 11, 12, 13\}$, whereas *Greedy* picks $\{1, 13\}$

Note that the algorithm will never choose N clusters, and it represents a significant improvement over ITF (recall Fig. 1.4.)

Collisions may be minimized by each head sending a schedule of when its neighbors should transmit that they are covered. Collisions are resolved by the same scheme as they were in *GLCA*. Acknowledgement schemes will also be similar for both algorithms.

3.9. Summary

Two Linked Cluster algorithms have been presented and analyzed. *GCLA* minimizes the number of clusters identically as *Greedy*. Thus this algorithm produces cluster organizations which will in general, lend themselves to standard solutions of the hidden terminal probem, such as Busy Tone. However it does not fare as well in the amount of communication required. Since this will affect the number of collisions in the network adversly, *GCLA* is probably best suited to environments in which the topologies are not highly dynamic. *TLCA* is well suited to more mobile situations since it has extremely low values of C and T, but clusters may not be minimized very well. Limitations of our work, and specific suggestions for further work are mentioned in the next chapter.

CHAPTER IV
CONCLUSION AND SUGGESTIONS FOR FURTHER WORK

4.1. Conclusion

Considerable insight has been gained in the difficulties associated with minimizing the number of clusters in stationless PRNS's, and plausible approaches to solve these problems have been suggested. We showed that 3 different formulations of the minimization problem are NP-complete. It was then shown that it is extremely unlikely that an efficient heuristic exists with constant bounded differential or fractional error. A simple greedy heuristic *Greedy* was analyzed extensively in terms of its worst-case fractional error, and it was shown that other, more complicated algorithms do not behave significantly better in the worst case. It was also proved that a famous bound by Vizing, for the cardinality of the minimum dominating set is also met by the dominating set selected by *Greedy*.

In the third chapter, we presented 2 distributed algorithms, one which produces the same dominating set as *Greedy*, but which entails considerable overhead in communication and is inefficient for dense graphs, and another in which the amount of communication is cut down significantly, but at the cost of not minimizing the number of clusters as well as *Greedy*. We do not claim that either of these algorithms could actually be implemented without modifications, some of which are discussed in the next section, but feel that some headway has been made on a difficult problem.

4.2. Suggestions for Further Work

In this section some of the limitations of the work in this thesis will be presented, along with suggestions for further research.

There are a number of interesting issues having to do with the centralized problem. First, we have given no results for worst case performance of heuristics for $CDSP$ and $SCDSP$. Second, an average case analysis of algorithms such as *Greedy*, which have the worst-case behaviour described in theorem 2.5, would help better understand how they actually do in practice. Third, all results mentioned in this thesis apply to the general case, in which the graph topology can be anything so long as it is connected. However, under certain classes of applications, it might be reasonable to assume that for example, all transmitting radii are equal, or that every radio is connected bidirectionally, to the same number of radios. Another possible restriction is to assume that all radios must be within a certain constrained area (say a square of side K miles.)

We have some very preliminary results for the case of special graph topologies, and mention them here solely for the benifit of the interested reader who wishes to gain some insight into these problems.

Lemma 4.1. *G is a representation of a PRN in which all transmitting radii are equal only if the following structures are not subgraphs of G: (Figure 4.1.)*

Proof: Fig. 4.1(a) can never be a subgraph because at most 5 non-intersecting circles of radius r can be drawn so that all their centers are contained in another circle of radius R.

Now suppose that 2 radios a, and b are both connected to 2 other nodes c, and d. It is easy to see that the distance between c and d is $< 2r$. This means that a third radio, e which is

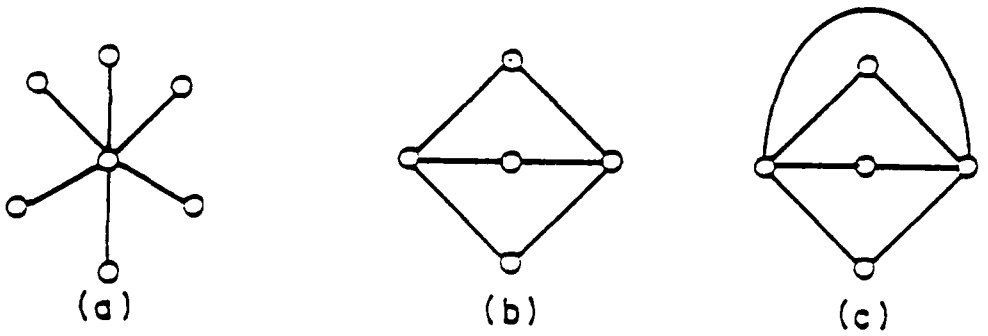


Fig. 4.1

shared by a and b must be connected to either c or to d.

Define the i_{min} to be the cardinality of the minimum cardinality maximal independent set of G.

Lemma 4.2. *$i_{min} = k_o$ for a graph G, if the following structure is not a subgraph:*

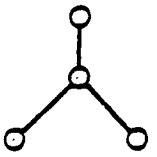


Fig. 4.2

Proof: Let D be a minimum cardinality dominating set, and suppose that it is not an independent set. Then let a and b be two adjacent nodes in D. Observe now that for every node in D there must be at least one node (including itself) that it covers exclusively i.e. no other member of D is adjacent to it. If this were not true then at least one node

could be removed from D without losing the dominating nature of the set. So let O_a be the set of nodes covered exclusively by node a. If $O_a = \{e\}$ then the set $D \cup \{e\} - \{a\}$ is a dominating set, and e is not adjacent to any other node in D. So let $|O_a| > 1$. Then let e and f be 2 nodes in O_a. Now observe that a is adjacent to b as well, and we know that it cannot be adjacent to 3 mutually non-adjacent nodes. It follows that e and f are neighbors. In fact since these nodes were arbitrarily picked from O_a, the members of this set must form a clique. Then it is easy to see that $D \cup \{e\} - \{a\}$ must also be a dominating set, and that none of the nodes in this set are adjacent to e (except itself). In this way we can keep replacing nodes of the dominating set until they are all mutually nonadjacent i.e. the set is independent From this we conclude that i_{min} is no bigger than K_o. Now observe that every (maximally) independent set is a dominating set. Thus $i_{min} \geq K_o$. The result follows.

We now turn to the work in Chapter 3. There are two major limitations of the algorithms presented. First, there are no global terminating conditions: i.e. a node has no way of knowing that the algorithm has terminated at all other nodes. Secondly, nodes are merely assigned to leader; clusters are not linked by gateway nodes. Thus our work on distributed algorithms can be improved by incorporating these features.

It is clear from this thesis that bad linked-cluster organizations can result in disasterous consequences. An interesting area of research is to examine the tradeoff of minimizing the number of clusters and of these consequences. We also argued that gateway nodes should be minimized. It would be interesting to explore the tradeoffs among the three problem formulations: *DSP*, *CDSP*, and *SCDSP* in terms of the most efficient linked-cluster organization. A better understanding of what multiaccess method is to be used once the clusters are set up, and of what the best routing strategy is, are prerequistes to attempting such an analysis, and we suggest work in these areas as well.

References

[1] R. E. Kahn, S.A. Gonemeyer, J. Burchfiel, R. C. Kunzelman, "Advances in Packet Radio Technology," *Proc. of the IEEE*, vol. 66, no. 10, pp. 1468–1496.

[2] R. E. Kahn, "The Organization of Computer Resources into a Packet Radio Network," *IEEE Transactions on Communications*, vol. COM–25, pp 169-178.

[3] R. Nelson, "Channel Access Protocols For Multi-Hop Broadcast Packet Radio Networks," *UCLA Computer Science Dept. Report*, CSD-820731.

[4] J. C. R. Licklider, A. Vezza, "Applications of Information Networks," *Proceedings of the IEEE*, vol. 66, no. 11.

[5] D. J. Baker, A. Ephridemes, "The Architectural Organization of a Packet Radio Network via a Distributed Algorithm," *IEEE Transactions on Communications*, vol. COM–29, pp. 1694–1701.

[6] F. A. Tobagi, L. Kleinrock, "Packet Switching in Radio Channels: Part II - The Hidden Terminal Problem and the Busy Tone Solution," *IEEE Transactions on Communications*, vol. COM–23, pp. 1417–1433.

[7] C. H. Papadimitriou, K. Steiglitz, *Combinatorial Optimization - Algorithms and Complexity*, Prentice Hall, 1982.

[8] G. Cornuejols, M. L. Fisher, G. l. Nemhauser, "Location of Bank Accounts to Optimize Float: An Analytic Study of Exact and Approximate Algorithms," *Management Science*, vol. 23, no. 8, pp. 789–810.

[9] A. M. Geoffrion, "Lagrangian Relaxation for Integer Programming," *Mathematical Programming Study*, vol. 2, pp. 82–114.

[10] D. S. Hochbaum, "Approximation Algorithms for the Set Covering and Vertex Covering Problems," *SIAM Journal of Computing*, vol. 11, pp. 555–556.

[11] V. Chvatal, "A Greedy Heuristic for the Set Covering Problem," *Mathematics of Operations Research*, vol. 4, pp. 233–235.

[12] D. S. Johnson, "Approximate Algorithms for Combinatorial Problems," *Journal of Computer System Science*, vol. 9, pp. 256–278.

[13] M. R. Garey, D. S. Johnson, *Computers and Intractability– A Guide to the Theory of NP-Completeness*, W. H. Freeman, 1979.

[14] R. G. Gallager, "Distributed Minimum Hop Algorithms," *MIT Laboratory for Information and Decision Systems Report*, P1175.

[15] C. Berge, *Graphs and Hypergraphs*, Dunod, Paris, 1970.

[16] R. G. Gallager, Private Communication.

[17] V. MacDonald, "The Cellular Concept," *Bell System Telecommunication Jounal*, vol. 58, No. 1.

[18] R. Karp, "Probabilistic Analysis of some Combinatorial Search Algorithms," in J. Traub (ed.), *Algorithms and Complexity: New Directions and Recent Results*, Academic Press, New York, 1976.

[19] R. Sinha, S. C. Gupta, "Mobile Packet Radio Networks– State of the Art," *IEEE Communications Magazine*, vol. 23, no. 3.

Distribution List

Defense Documentation Center — 12 Copies
Cameron Station
Alexandria, Virginia 22314

Assistant Chief for Technology — 1 Copy
Office of Naval Research, Code 200
Arlington, Virginia 22217

Office of Naval Research — 2 Copies
Information Systems Program
Code 437
Arlington, Virginia 22217

Office of Naval Research — 1 Copy
Branch Office, Boston
495 Summer Street
Boston, Massachusetts 02210

Office of Naval Research — 1 Copy
Branch Office, Chicago
536 South Clark Street
Chicago, Illinois 60605

Office of Naval Research — 1 Copy
Branch Office, Pasadena
1030 East Greet Street
Pasadena, California 91106

Naval Research Laboratory — 6 Copies
Technical Information Division, Code 2627
Washington, D.C. 20375

Dr. A. L. Slafkosky — 1 Copy
Scientific Advisor
Commandant of the Marine Corps (Code RD-1)
Washington, D.C. 20380

Office of Naval Research
Code 455
Arlington, Virginia 22217 — 1 Copy

Office of Naval Research
Code 458
Arlington, Virginia 22217 — 1 Copy

Naval Electronics Laboratory Center
Advanced Software Technology Division
Code 5200
San Diego, California 92152 — 1 Copy

Mr. E. H. Gleissner
Naval Ship Research & Development Center
Computation and Mathematics Department
Bethesda, Maryland 20084 — 1 Copy

Captain Grace M. Hopper
Naval Data Automation Command
Code 00H
Washington Navy Yard
Washington, DC 20374 — 1 Copy

Advanced Research Projects Agency
Information Processing Techniques
1400 Wilson Boulevard
Arlington, Virginia 22209 — 1 Copy

Dr. Stuart L. Brodsky
Office of Naval Research
Code 432
Arlington, Virginia 22217 — 1 Copy

Prof. Fouad A. Tobagi
Computer Systems Laboratory
Stanford Electronics Laboratories
Department of Electrical Engineering
Stanford University
Stanford, CA 94305